Лабораторная работа №2

**Цель работы:**

Познакомиться с основой объектного подхода в языке C#, механизмом наследования**.**

**Необходимые теоретические сведения**

**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:*

```
public class Animal
{
    public void Greet()
    {
        Console.WriteLine("Hello, I'm some sort of animal!");
    }
}

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’).

C# и .NET поддерживают только *одиночное наследование*. Это означает, что каждый класс может наследовать члены только одного класса. Но зато поддерживается транзитивное наследование, которое позволяет определить иерархию наследования для набора типов. Другими словами, тип D может наследовать возможности типа C, который в свою очередь наследует от типа B, который наследует от базового класса A. Благодаря транзитивности наследования члены типа A будут доступны для типа D.

Не все члены базового класса наследуются производными классами. Следующие члены не наследуются.

- Статические конструкторы, которые инициализируют статические данные класса.
- Конструкторы экземпляров, которые вызываются для создания нового экземпляра класса. Каждый класс должен определять собственные конструкторы.

- Методы завершения, которые вызываются сборщиком мусора среды выполнения для уничтожения экземпляров класса.

**Доступность родительских элементов**

- Закрытые члены являются видимыми только в производных классах, которые вложены в базовый класс. Для других производных классов они невидимы. В следующем примере класс A.B является вложенным и производным от A, а C является производным от A. Закрытое поле A.value является видимым в классе A.B. Но если раскомментировать строки метода C.GetValue, то при компиляции этого примера возникнет ошибка CS0122: "'A.value' недоступен из-за его уровня защиты".

```
using System;

public class A
{
   private int value = 10;

   public class B : A
   {
       public int GetValue()
       {
           return this.value;
       }
   }
}

public class C : A
{
//    public int GetValue()
//    {
//        return this.value;
//    }
}

public class Example
{
    public static void Main(string[] args)
    {
        var b = new A.B();
        Console.WriteLine(b.GetValue());
    }
}
// The example displays the following output:
//       10
```

- Защищенные члены являются видимыми только в производных классах.
- Внутренние члены являются видимыми только в производных классах, которые находятся в той же сборке, что и базовый класс. Они не будут видимыми в производных классах, расположенных в других сборках.
- Открытые члены являются видимыми в производных классах, а также входят в общедоступный интерфейс производных классов. Унаследованные открытые

члены можно вызывать так же, как если бы они были определены в самом производном классе. В следующем примере класс A определяет метод с именем Method1, а класс B наследует от класса A. В нашем примере Method1 вызывается так, как если бы это был метод класса B.

```
public class A
{
    public void Method1()
    {
        // Method implementation.
    }
}

public class B : A
{ }

public class Example
{
    public static void Main()
    {
        B b = new B();
        b.Method1();
    }
}
```

**Задание**

Написать C# программу на основе лабораторной №1 реализующую работу с классами по вариантам.

Каждый из классов должен удовлетворять следующим требованиям:

1. Все классы не должны содержать одинаковых не наследуемых свойств.
2. Конструкторы должны быть определены через конструкторы родительских классов.
3. Конструктор от экземпляра такого же класса.
4. Метод ToString позволяющий преобразовать всю информацию о классе в строку.
5. Метод Compare позволяющий сравнить на равенство с экземпляром такого же класса.

Разработанная программа должна быть консольной, позволяющей на выбор добавлять объекты классов, при наличии точно такого же экземпляра сообщать об этом пользователю. Кроме того должна быть возможность просмотреть все добавленные объекты заданного класса.

В отчёт обязательно построить диаграмму классов, описывающую иерархию созданных классов.

**Варианты**:

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, двигатель внутреннего сгорания, дизель, реактивный двигатель
13. Республика, монархия, королевство, государство
14. Млекопитающее, парнокопытное, птица, животное
15. Корабль, пароход, парусник, корвет