Лабораторная работа №10

Тема: Разработка программ, реализующих механизм наследования классов.

Цель: Научиться разработке программ, реализующих механизм наследования классов.

Теоретические сведения:

Индексаторы — это члены класса, которые позволяют получить доступ к полям класса способом доступа к элементам массива, т.е. по индексу.

Индексаторы очень похожи на свойства. Для каждого индексатора так же может быть два метода: один — для чтения значения поля (**get**), другой — для записи в него какого-то значения (**set**). В индексаторе могут быть определены оба метода или только какой-либо один.

Пример 1. Разработаем класс для работы с одномерным массивом. Создадим в нём индексатор, позволяющий обращаться к элементу массива по индексу.

Текст программы:

```
using System;
namespace Test_Indeksator1
{
class Mas
{
private int n;
private double []x;
public Mas()
{
n=1;
x=new double[n];
x[0]=1;
}
public Mas(int n0)
{
n=n0;
if(n<1) n=1;
x=new double[n];
for (int i = 0; i < n; i++) {
x[i]=1+i;
}
```

```
}
public double this[int i]
        {
set
{
x[i]=value;
}
get
{
return x[i];
}
}
public void print()
{
Console.WriteLine("Массив:");
for (int i = 0; i < n; i++) {
Console.WriteLine(x[i]);
}
}
}
class Program
{
public static void Main(string[] args)
{
Console.WriteLine("Тестирование работы индексаторов");
Mas x = new Mas(3);
// Использование индексатора для записи
x[0]=5;
// Использование индексатора для чтения
Console.WriteLine(x[1]);
x.print();
Console.Write("Press any key to continue . .
. ");
Console.ReadKey(true);
}
}
}
```

Результат работы программы:

```
Тестирование работы индексаторов
2
Массив:
5
2
3
Press any key to continue . . .
```

Рисунок 1 – Выполнение программы

Пример 2. Разработаем класс для работы с двумерным массивом (матрицей). Создадим в нём индексатор, позволяющий обращаться к элементу матрицы по индексу.

Текст программы:

```
using System;
namespace Test_Indeksator2
{
	class Matr
{
private int n;
private int m;
private double [,]x;
public Matr()
{
n=m=1;
x=new double[n,m];
x[0,0]=1;
}
public Matr(int n0, int m0)
{
n=n0;
if(n<1) n=1;
m=m0;
if(m<1) m=1;
x=new double[n,m];
for (int i = 0; i < n; i++) {
```

```
for (int j = 0; j < m; j++) {
x[i,j]=(i+1)*10+j+1;
}
}
}
public double this[int i, int j]
{
set
{
x[i,j]=value;
}
get
{
return x[i,j];
}
}
public void print()
{
Console.WriteLine("Матрица:");
for (int i = 0; i < n; i++) {
for (int j = 0; j < m; j++) {
Console.Write(x[i,j].ToString()+" ");
}
Console.WriteLine();
}
}
}
class Program
{
public static void Main(string[] args)
{
Console.WriteLine("Тестирование работы
индексаторов");
Matr x = new Matr(3,3);
        // использование индексатора для записи
x[0,0]=5;
// использование индексатора для чтения
```

```
Console.WriteLine(x[1,1]);
x.print();
Console.Write("Press any key to continue . .
. ");
Console.ReadKey(true);
}
}
}
```

Результат работы программы:

```
Тестирование работы индексаторов
22
Матрица:
5 12 13
21 22 23
31 32 33
Press any key to continue . . .
```

Рисунок 2 – Выполнение работы программы

Абстрактные классы

Иногда требуется создать базовый класс, в котором определяется лишь самая общая форма для всех его производных классов, а наполнение ее деталями предоставляется каждому из этих классов. В таком классе определяется лишь характер методов, которые должны быть конкретно реализованы в производных классах, а не в самом базовом классе. Подобная ситуация возникает, например, в связи с невозможностью получить содержательную реализацию метода в базовом классе.

Создавая собственные библиотеки классов, вы можете сами убедиться в том, что у метода зачастую отсутствует содержательное определение в контексте его базового класса. Подобная ситуация разрешается двумя способами. Один из них состоит в том, чтобы просто выдать предупреждающее сообщение. Такой способ может пригодиться в определенных ситуациях, например при отладке, но в практике программирования он обычно не применяется. В подобных случаях требуется какой-то способ, гарантирующий, что в производном классе действительно будут переопределены все необходимые методы. И такой способ в C# имеется. Он состоит в использовании абстрактного метода.

Абстрактный метод создается с помощью указываемого модификатора типа **abstract**. У абстрактного метода отсутствует тело, и поэтому он не реализуется в базовом классе. Это означает, что он должен быть переопределен в производном классе, поскольку его вариант из базового класса просто непригоден для использования. Нетрудно догадаться, что абстрактный метод автоматически становится виртуальным и не требует указания модификатора virtual. В действительности совместное использование модификаторов virtual и abstract считается ошибкой. Для определения абстрактного метода служит приведенная ниже общая форма:

abstract тип имя(список_параметров);

Как видите, у абстрактного метода отсутствует тело. Модификатор abstract может применяться только в методах экземпляра, но не в статических методах (static). Абстрактными могут быть также индексаторы и свойства.

Класс, содержащий один или больше абстрактных методов, должен быть также объявлен как абстрактный, и для этого перед его объявлением class указывается модификатор abstract. А поскольку реализация абстрактного класса не определяется полностью, то у него не может быть объектов. Следовательно, попытка создать объект абстрактного класса с помощью оператора new приведет к ошибке во время компиляции!

Когда производный класс наследует абстрактный класс, в нем должны быть реализованы все абстрактные методы базового класса. В противном случае производный класс должен быть также определен как abstract. Таким образом, атрибут abstract наследуется до тех пор, пока не будет достигнута полная реализация класса.

Пример 3. Использование абстрактного класса.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   // Создаем абстрактный класс
   abstract class UserInfo
   {
      protected string Name;
      protected byte Age;
```

```
    public UserInfo(string Name, byte Age)
    {
        this.Name = Name;
        this.Age = Age;
    }

    // Абстрактный метод
    public abstract string ui();
}

class UserFamily : UserInfo
{
    string Family;

    public UserFamily(string Family, string Name, byte Age) : base (Name,
Age)
    {
        this.Family = Family;
    }

    // Переопределяем метод ui
    public override string ui()
    {
        return Family + " " + Name + " " + Age;
    }
}

class Program
{
    static void Main(string[] args)
    {
        UserFamily user1 = new UserFamily("Erohin", "Alexandr", 26);
        Console.WriteLine(user1.ui());

        Console.ReadLine();
    }
}
```

```
}
```

В данном примере создается абстрактный класс UserInfo в котором инкапсулируется абстрактный метод ui(), который, в свою очередь, переопределяется в классе UserFamily.

Следует отметить, что в абстрактные классы вполне допускается (и часто практикуется) включать конкретные методы, которые могут быть использованы в своем исходном виде в производном классе. А переопределению в производных классах подлежат только те методы, которые объявлены как abstract.

Основы наследования

Наследование является одним из трех основополагающих принципов объектно-ориентированного программирования, поскольку оно допускает создание иерархических классификаций. Благодаря наследованию можно создать общий класс, в котором определяются характерные особенности, присущие множеству связанных элементов. От этого класса могут затем наследовать другие, более конкретные классы, добавляя в него свои индивидуальные особенности.

В языке C# класс, который наследуется, называется базовым, а класс, который наследует, — производным. Следовательно, производный класс представляет собой специализированный вариант базового класса. Он наследует все переменные, методы, свойства и индексаторы, определяемые в базовом классе, добавляя к ним свои собственные элементы.

Поддержка наследования в C# состоит в том, что в объявление одного класса разрешается вводить другой класс. Для этого при объявлении производного класса указывается базовый класс. При установке между классами отношения "является" строится зависимость между двумя или более типами классов. Базовая идея, лежащая в основе классического наследования, заключается в том, что новые классы могут создаваться с использованием существующих классов в качестве отправной точки:

Пример 4. Использование наследования

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
```

```
class ProfessorWeb
{
   const string ADDR = "http:\\professorweb.ru";
   public string level, inLevel;
   public int numberSt;
   private string inf;

   public void InfoPW()
   {
      Console.WriteLine("Сайт: {0}\nРаздел: {1}\nПодраздел: {2}\nКол-
во статей:{3}",ADDR,level,inLevel,numberSt);
   }
}

// Объявляем класс, унаследованный от класса ProfessorWeb
class CSharp : ProfessorWeb
{
   public string st;

   // Поля класса ProfessorWeb доступны через конструктор
наследуемого класса
   public CSharp(string level, string inLevel, int numberSt, string st)
   {
      this.level = level;
      this.inLevel = inLevel;
      this.numberSt = numberSt;
      this.st = st;
   }

   public void StWrite()
   {
      Console.WriteLine("Статья: "+st);
   }
}

class Program
{
```

```
        static void Main()
        {
            CSharp obj = new CSharp(level: "C#", inLevel: "Перегрузка", numberSt: 7, st: "Перегрузка методов");
            obj.InfoPW();
            obj.StWrite();

            Console.ReadLine();
        }
    }
}
```

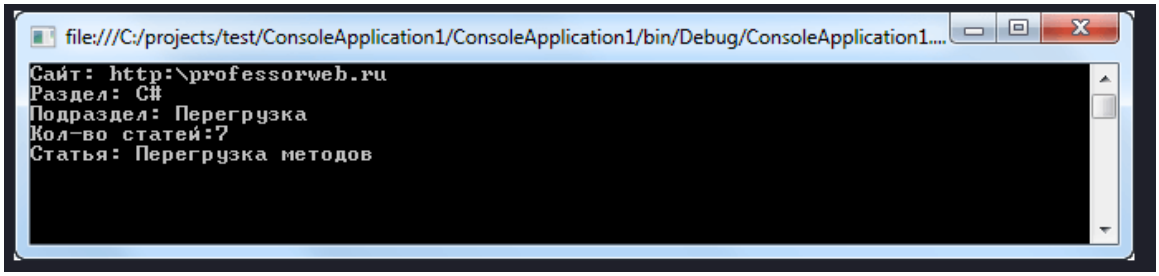

Рисунок 3 – Выполнение программы

Всякий раз, когда один класс наследует от другого, после имени производного класса указывается имя базового класса, отделяемое двоеточием. В C# синтаксис наследования класса удивительно прост и удобен в использовании. Ниже представлено схематичное представление класса CSharp из вышеуказанного примера:

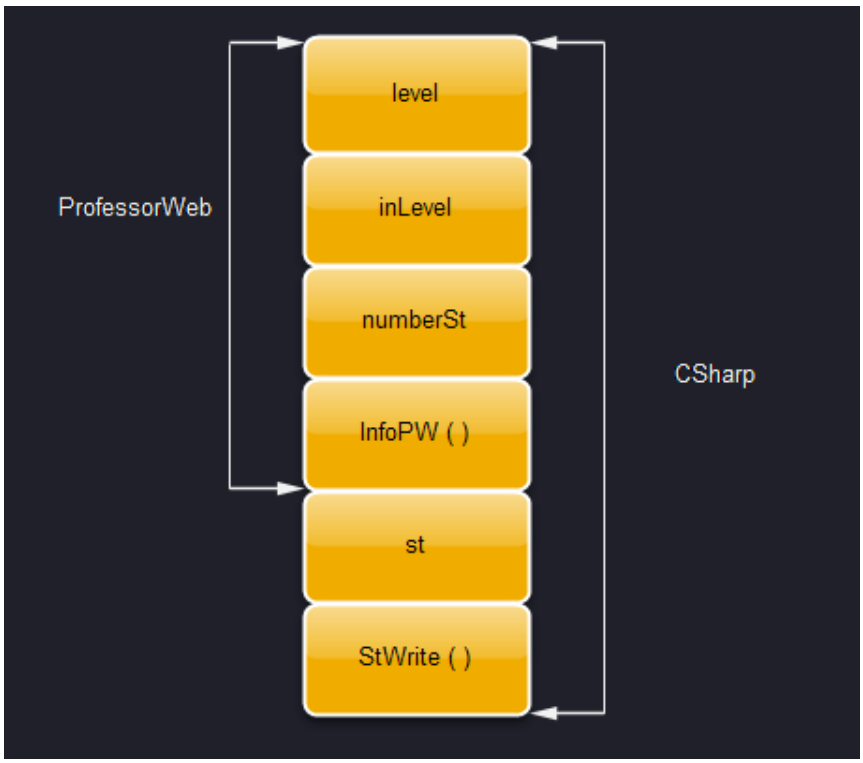

Рисунок 4 – Схема наследования

Как видно класс CSharp получает доступ к полям и методам класса ProfessorWeb. Всегда помните, что наследование предохраняет инкапсуляцию, а потому приватные члены никогда не могут быть доступны через ссылку на объект. Т.е. поле inf из примера не может быть доступно для вызова с помощью экземпляра класса obj.

Для любого производного класса можно указать только один базовый класс. В C# не предусмотрено наследование нескольких базовых классов в одном производном классе. (В этом отношении C# отличается от C++, где допускается наследование нескольких базовых классов. Данное обстоятельство следует принимать во внимание при переносе кода C++ в C#.) Тем не менее можно создать иерархию наследования, в которой производный класс становится базовым для другого производного класса. (Разумеется, ни один из классов не может быть базовым для самого себя как непосредственно, так и косвенно.) Но в любом случае производный класс наследует все члены своего базового класса, в том числе переменные экземпляра, методы, свойства и индексаторы.

Главное преимущество наследования заключается в следующем: как только будет создан базовый класс, в котором определены общие для множества объектов атрибуты, он может быть использован для создания любого числа более конкретных производных классов. А в каждом производном классе может быть точно выстроена своя собственная классификация.

Конструкторы и наследование

В иерархии классов допускается, чтобы у базовых и производных классов были свои собственные конструкторы. В связи с этим возникает следующий резонный вопрос: какой конструктор отвечает за построение объекта производного класса: конструктор базового класса, конструктор производного класса или же оба? На этот вопрос можно ответить так: конструктор базового класса конструирует базовую часть объекта, а конструктор производного класса — производную часть этого объекта. И в этом есть своя логика, поскольку базовому классу неизвестны и недоступны любые элементы производного класса, а значит, их конструирование должно происходить раздельно.

Если конструктор определен только в производном классе, то все происходит очень просто: конструируется объект производного класса, а базовая часть объекта автоматически собирается его конструктором, используемым по умолчанию.

Когда конструкторы определяются как в базовом, так и в производном классе, процесс построения объекта несколько усложняется, поскольку должны выполняться конструкторы обоих классов. В данном случае приходится обращаться к ***ключевому слову base***, которое находит двоякое применение: во-первых, для вызова конструктора базового класса; и во-вторых, для доступа к члену базового класса, скрывающегося за членом производного класса.

С помощью формы расширенного объявления конструктора производного класса и ключевого слова base в производном классе может быть вызван конструктор, определенный в его базовом классе. Ниже приведена общая форма этого расширенного объявления:

конструктор_производного_класса(список_параметров) : base (список_аргументов)

{

// тело конструктора

}

где *список_аргументов* обозначает любые аргументы, необходимые конструктору в базовом классе. Обратите внимание на местоположение двоеточия.

Пример 5. Использование конструктора при наследовании

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace ConsoleApplication1
{
   class MyClass
   {
      public int x, y, z;

      // Конструктор базового класса
      public MyClass(int x, int y, int z)
      {
         this.x = x;
         this.y = y;
         this.z = z;
      }
   }

   class ClassA : MyClass
   {
      int point;

      // Конструктор производного класса
      public ClassA(int point, int x, int y, int z)
         : base(x, y, z)
      {
         this.point = point;
      }

      public void Pointer(ClassA obj)
      {
         obj.x *= obj.point;
         obj.y *= obj.point;
         obj.z *= obj.point;
         Console.WriteLine("Новые координаты объекта: {0} {1} {2}", obj.x,
obj.y, obj.z);
      }
   }
```

```
        class Program
        {
            static void Main()
            {
                ClassA obj = new ClassA(10, 1, 4, 3);
                Console.WriteLine("Координаты объекта: {0} {1} {2}", obj.x, obj.y,
obj.z);
                obj.Pointer(obj);

                Console.ReadLine();
            }
        }
    }
```

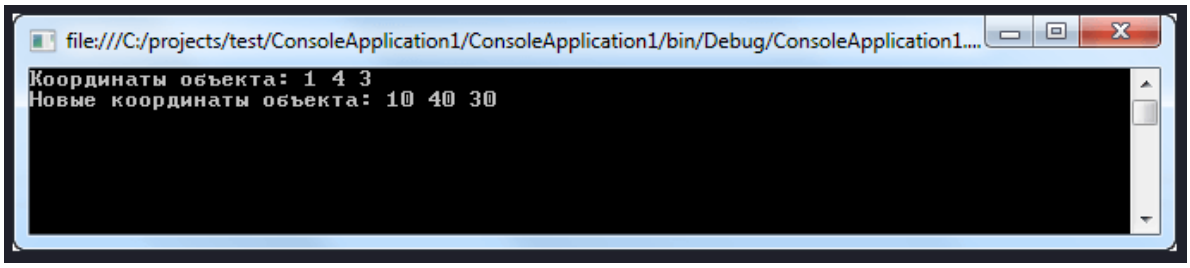

Рисунок 5 – Выполнение программы

С помощью ключевого слова base можно вызвать конструктор любой формы, определяемой в базовом классе, причем выполняться будет лишь тот конструктор, параметры которого соответствуют переданным аргументам.

А теперь рассмотрим вкратце основные принципы действия ключевого слова base. Когда в производном классе указывается ключевое слово base, вызывается конструктор из его непосредственного базового класса. Следовательно, ключевое слово base всегда обращается к базовому классу, стоящему в иерархии непосредственно над вызывающим классом. Это справедливо даже для многоуровневой иерархии классов. Аргументы передаются базовому конструктору в качестве аргументов метода base(). Если же ключевое слово отсутствует, то автоматически вызывается конструктор, используемый в базовом классе по умолчанию.

Ссылки на базовый класс и объекты производных классов

C# является строго типизированным языком программирования. Помимо стандартных преобразований и автоматического продвижения простых типов значений, в этом языке строго соблюдается принцип

совместимости типов. Это означает, что переменная ссылки на объект класса одного типа, как правило, не может ссылаться на объект класса другого типа.

Вообще говоря, переменная ссылки на объект может ссылаться только на объект своего типа.

Но из этого принципа строгого соблюдения типов в C# имеется одно важное исключение: переменной ссылки на объект базового класса может быть присвоена ссылка на объект любого производного от него класса. Такое присваивание считается вполне допустимым, поскольку экземпляр объекта производного типа инкапсулирует экземпляр объекта базового типа. Следовательно, по ссылке на объект базового класса можно обращаться к объекту производного класса:

Пример 6. Использование базового класса.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class ClassA
   {
      public int a;

      public ClassA(int i)
      {
         a = i;
      }
   }

   class ClassB : ClassA
   {
      public int b;

      public ClassB(int r, int i)
         : base(i)
      {
         b = r;
```

```
    }
}

class ClassC
{
    public int a;

    public ClassC(int i)
    {
        a = i;
    }
}

class Program
{
    static void Main()
    {
        ClassA objA1 = new ClassA(10);
        ClassA objA2 = new ClassA(5);

        // Присваивание однотипных объектов правильно:
        objA2 = objA1;

        ClassB objB = new ClassB(8, 9);
        // Присваивание объекта, имеющего тип унаследованного класса
допускается:
        objA1 = objB;
        // Так нельзя
        // objB = objA1;

        // Присваивать похожие, но не связанные друг с другом
        // принципом наследования объекты нельзя
        ClassC objC = new ClassC(8);
        // objC = objA1;

        Console.ReadLine();
    }
```

```
    }
}
```

Следует особо подчеркнуть, что доступ к конкретным членам класса определяется типом переменной ссылки на объект, а не типом объекта, на который она ссылается. Это означает, что если ссылка на объект производного класса присваивается переменной ссылки на объект базового класса, то доступ разрешается только к тем частям этого объекта, которые определяются базовым классом. И в этом есть своя логика, поскольку базовому классу ничего не известно о тех членах, которые добавлены в производный от него класс.

Один из самых важных моментов для присваивания ссылок на объекты производного класса переменным базового класса наступает тогда, когда конструкторы вызываются в иерархии классов. Как вам должно быть уже известно, в классе нередко определяется конструктор, принимающий объект своего класса в качестве параметра. Благодаря этому в классе может быть сконструирована копия его объекта.

Виртуальные методы, свойства и индексаторы

Полиморфизм предоставляет подклассу способ определения собственной версии метода, определенного в его базовом классе, с использованием процесса, который называется *переопределением метода (method overriding)*. Чтобы пересмотреть текущий дизайн, нужно понять значение ключевых слов virtual и override.

Виртуальным называется такой метод, который объявляется как **virtual** в базовом классе. Виртуальный метод отличается тем, что он может быть переопределен в одном или нескольких производных классах. Следовательно, у каждого производного класса может быть свой вариант виртуального метода. Кроме того, виртуальные методы интересны тем, что именно происходит при их вызове по ссылке на базовый класс. В этом случае средствами языка C# определяется именно тот вариант виртуального метода, который следует вызывать, исходя из типа объекта, к которому происходит обращение по ссылке, причем это делается во время выполнения. Поэтому при ссылке на разные типы объектов выполняются разные варианты виртуального метода. Иными словами, вариант выполняемого виртуального метода выбирается по типу объекта, а не по типу ссылки на этот объект.

Так, если базовый класс содержит виртуальный метод и от него получены производные классы, то при обращении к разным типам объектов по ссылке на базовый класс выполняются разные варианты этого виртуального метода.

Метод объявляется как виртуальный в базовом классе с помощью ключевого слова virtual, указываемого перед его именем. Когда же виртуальный метод переопределяется в производном классе, то для этого используется модификатор **override**. А сам процесс повторного определения виртуального метода в производном классе называется переопределением метода. При переопределении метода - имя, возвращаемый тип и сигнатура переопределяющего метода должны быть точно такими же, как и у того виртуального метода, который переопределяется. Кроме того, виртуальный метод не может быть объявлен как static или abstract.

Переопределение метода служит основанием для воплощения одного из самых эффективных в C# принципов: ***динамической диспетчеризации методов***, которая представляет собой механизм разрешения вызова во время выполнения, а не компиляции. Значение динамической диспетчеризации методов состоит в том, что именно благодаря ей в C# реализуется динамический полиморфизм.

Если при наличии многоуровневой иерархии виртуальный метод не переопределяется в производном классе, то выполняется ближайший его вариант, обнаруживаемый вверх по иерархии.

И еще одно замечание: свойства также подлежат модификации ключевым словом virtual и переопределению ключевым словом override. Это же относится и к индексаторам.

Пример 7. Использование виртуальных методов, свойств и индексаторов

```
// Реализуем класс содержащий информацию о шрифтах
// и использующий виртуальные методы, свойства и индексаторы

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
  // Базовый класс
  class Font
  {
    string TypeFont;
    short FontSize;
```

```
public Font()
{
   TypeFont = "Arial";
   FontSize = 12;
}

public Font(string TypeFont, short FontSize)
{
   this.TypeFont = TypeFont;
   this.FontSize = FontSize;
}

public string typeFont
{
   get
   {
      return TypeFont;
   }

   set
   {
      TypeFont = value;
   }
}

public short fontSize
{
   get
   {
      return FontSize;
   }

   set
   {
      FontSize = value;
   }
```

```
    }

    // Создаем виртуальный метод
    public virtual string FontInfo(Font obj)
    {
      string s = "Информация о шрифте: \n----------\n\n" +
        "Тип шрифта: " + typeFont +
        "\nРазмер шрифта: " + fontSize + "\n";
      return s;
    }
  }

  // Производный класс 1 уровня
  class ColorFont : Font
  {
    byte Color;

    public ColorFont(byte Color, string TypeFont, short FontSize)
      : base(TypeFont, FontSize)
    {
      this.Color = Color;
    }

    // Переопределение для виртуального метода
    public override string FontInfo(Font obj)
    {
      // Используется ссылка на метод, определенный в базовом классе Font
      return base.FontInfo(obj) + "Цвет шрифта: " + Color + "\n";
    }

    // Создадим виртуальное свойство
    public virtual byte color
    {
      set
      {
        Color = value;
```

```
        }

        get
        {
            return Color;
        }
    }
}

// Производный класс 2 уровня
class GradientColorFont : ColorFont
{
    char TypeGradient;

    public GradientColorFont(char TypeGradient, byte Color, string TypeFont, short FontSize) :
        base(Color, TypeFont, FontSize)
    {
        this.TypeGradient = TypeGradient;
    }

    // Опять переопределяем виртуальный метод
    public override string FontInfo(Font obj)
    {
        // Используется ссылка на метод определенный в производном классе FontColor
        return base.FontInfo(obj) + "Тип градиента: " + TypeGradient + "\n\n";
    }

    // Переопределим виртуальное свойство
    public override byte color
    {
        get
        {
            return base.color;
        }
```

```
            set
            {
               if (value < 10)
                  base.color = 0;
               else
                  base.color = (byte)(value - 0x0A);
            }
         }
      }

      // Еще один производный класс 1 уровня
      class FontStyle : Font
      {
         string style;

         public FontStyle(string style, string TypeFont, short FontSize) : base
(TypeFont, FontSize)
         {
            this.style = style;
         }

         // Данный класс не переопределяет виртуальный метод
         // поэтому при вызове метода FontInfo ()
         // вызывается метод созданный в базовом классе
      }

      class Program
      {
         static void Main()
         {
            ColorFont font1 = new ColorFont(Color: 0xCF, TypeFont: "MS
Trebuchet", FontSize: 16);
            Console.WriteLine(font1.FontInfo(font1));

            GradientColorFont font2 = new GradientColorFont(Color: 0xFF,
TypeFont: "Times New Roman", FontSize: 10, TypeGradient: 'R');
            Console.WriteLine(font2.FontInfo(font2));
```

```
            font2.color = 0x2F;
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine("Видоизмененный цвет font2");
            Console.ForegroundColor = ConsoleColor.Gray;
            Console.WriteLine(font2.FontInfo(font2));

            FontStyle font3 = new FontStyle(style: "oblique", TypeFont: "Calibri",
FontSize: 16);
            Console.WriteLine(font3.FontInfo(font3));

            Console.ReadLine();
        }
    }
}
```

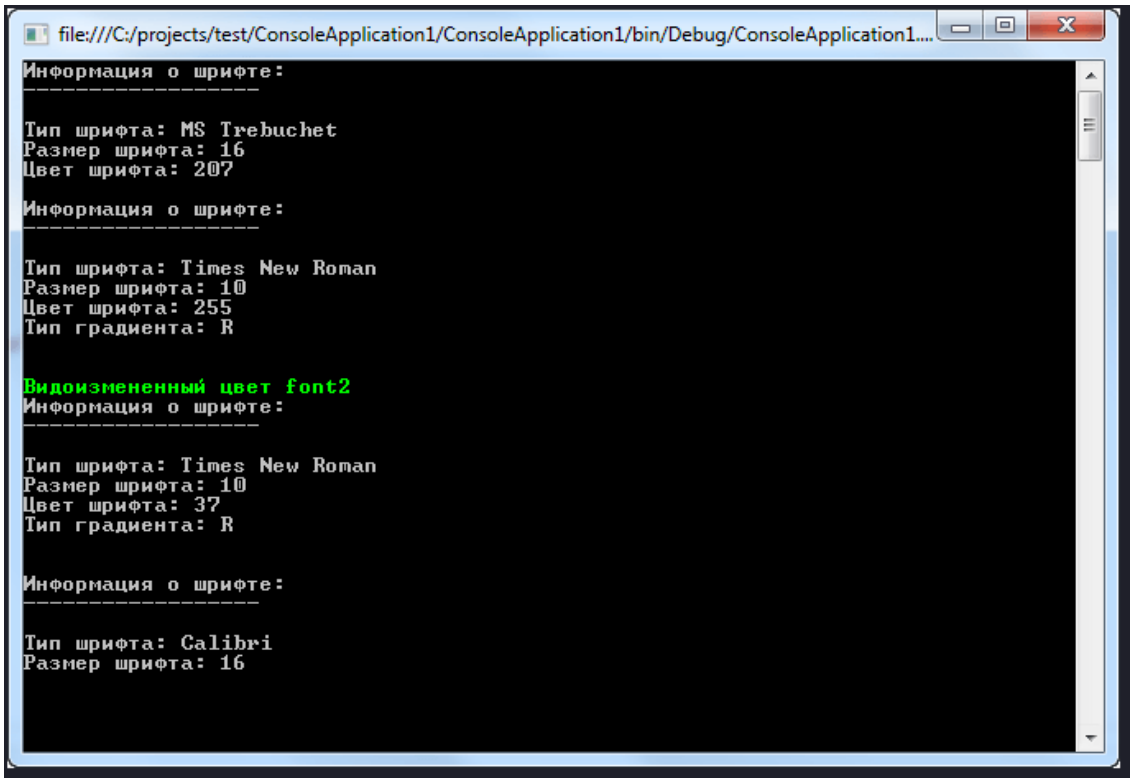

Рисунок 6 – Выполнение программы

Давайте рассмотрим данный пример более подробно. В базовом классе Font инкапсулируется виртуальный метод FontInfo (), возвращающий информацию о шрифте. В производном классе FontColor данный метод переопределяется с помощью ключевого слова override, поэтому при создании экземпляра данного класса и вызова метода FontInfo() в исходную

информацию возвращается помимо первоначальных данных еще и цвет шрифта. Затем данный метод вновь переопределяется в классе GradientColorFont, унаследованном от класса FontColor. Обратите внимание, что здесь переопределяется не исходный метод базового класса Font, а уже переопределенный метод класса FontColor. В этом и заключается принцип динамического полиморфизма!

Так же обратите внимание, что в данном примере используется виртуальное свойство color, принцип использования которого аналогичен использованию виртуального метода.

Выполнение работы:

Каждый разрабатываемый класс должен, как правило, содержать следующие элементы: скрытые поля, конструкторы с параметрами и без параметров, методы; свойства, индексаторы; перегруженные операции. Функциональные элементы класса должны обеспечивать непротиворечивый, полный, минимальный и удобный интерфейс класса.

В программе должна выполняться проверка всех разработанных элементов класса.

В отчете необходимо изобразить блок-схему алгоритма программы!!!

Вариант 1

Описать класс для работы с одномерным массивом целых чисел (вектором). Обеспечить следующие возможности:

• задание произвольных целых границ индексов при создании объекта;

• обращение к отдельному элементу массива с контролем выхода за пределы массива;

• выполнение операций поэлементного сложения и вычитания массивов с одинаковыми границами индексов;

• выполнение операций умножения и деления всех элементов массива на скаляр;

• вывод на экран элемента массива по заданному индексу и всего массива.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 2

Описать класс для работы с одномерным массивом строк фиксированной длины. Обеспечить следующие возможности:

• задание произвольных целых границ индексов при создании объекта;

• обращение к отдельной строке массива по индексу с контролем выхода за пределы массива;

• выполнение операций поэлементного сцепления двух массивов с образованием нового массива;

• выполнение операций слияния двух массивов с исключением повторяющихся элементов;

• вывод на экран элемента массива по заданному индексу и всего массива.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 3

Описать класс многочленов от одной переменной, задаваемых степенью многочлена и массивом коэффициентов. Обеспечить следующие возможности:

• вычисление значения многочлена для заданного аргумента;

• операции сложения, вычитания и умножения многочленов с получением нового объекта-многочлена;

• получение коэффициента, заданного по индексу;

• вывод на экран описания многочлена.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 4

Описать класс, обеспечивающий представление матрицы произвольного размера с возможностью изменения числа строк и столбцов, вывода на экран подматрицы любого размера и всей матрицы, доступа по индексам к элементу матрицы.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 5

Описать класс для работы с восьмеричным числом, хранящимся в виде строки символов. Реализовать конструкторы, свойства, методы и следующие операции:

• операции присваивания, реализующие значимую семантику;

• операции сравнения;

• преобразование в десятичное число;

• форматный вывод;

• доступ к заданной цифре числа по индексу.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 6

Описать класс «домашняя библиотека». Предусмотреть возможность работы с произвольным числом книг, поиска книги по какому-либо признаку (по автору, по году издания или категории), добавления книг в библиотеку, удаления книг из нее, доступа к книге по номеру.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 7

Написать класс «записная книжка». Предусмотреть возможность работы с произвольным числом записей, поиска записи по какому-либо признаку (например, по фамилии, дате рождения или номеру телефона), добавления и удаления записей, сортировки по фамилии и доступа к записи по номеру.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 8

Написать класс «студенческая группа». Предусмотреть возможность работы с переменным числом студентов, поиска студента по какому-либо признаку (например, по фамилии, имени, дате рождения), добавления и удаления записей, сортировки по разным полям, доступа к записи по номеру.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 9

Написать класс, реализующий тип данных «вещественная матрица» и работу с ними. Класс должен реализовывать следующие операции над матрицами:

• сложение, вычитание (как с другой матрицей, так и с числом);

• комбинированные операции присваивания (+=, -=);

• операции сравнения на равенство/неравенство;

• операции вычисления обратной и транспонированной матрицы;

• доступ к элементу по индексам.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 10

Описать класс, реализующий тип данных «вещественная матрица» и работу с ними. Класс должен реализовывать следующие операции над матрицами:

• умножение, деление (как на другую матрицу, так и на число);

• комбинированные операции присваивания (*=, /=);

• операцию возведения в степень;

• методы вычисления детерминанта и нормы;

• доступ к элементу по индексам.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 11

Описать класс, реализующий тип данных «вещественная матрица» и работу с ними. Класс должен реализовывать следующие операции над матрицами:

• методы, реализующие проверку типа матрицы (квадратная, диагональная, нулевая, единичная, симметричная, верхняя треугольная, нижняя треугольная);

• операции сравнения на равенство/неравенство;

• доступ к элементу по индексам.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 12

Описать класс «множество», позволяющий выполнять основные операции: добавление и удаление элемента, пересечение, объединение и разность множеств.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 13

Описать класс «предметный указатель». Каждый компонент указателя содержит слово и номера страниц, на которых, это слово встречается. Количество номеров страниц, относящихся к одному слову, от одного до десяти. Предусмотреть возможность формирования указателя с клавиатуры и из файла, вывода указателя, вывода номеров страниц для заданного слова, удаления элемента из указателя.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 14

Описать класс «автостоянка» для хранения сведений об автомобилях. Для каждого автомобиля записываются госномер, цвет, фамилия владельца и признак присутствия на стоянке. Обеспечить возможность поиска автомобиля по разным критериям, вывода списка присутствующих и отсутствующих на стоянке автомобилей, доступа к имеющимся сведениям по номеру места.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 15

Описать класс «колода карт», включающий закрытый массив элементов класса «карта». В карте хранятся масть и номер. Обеспечить возможность вывода карты по номеру, вывода всех карт, перемешивания колоды и выдачи всех карт из колоды поодиночке и по 6 штук в случайном порядке.

Написать программу, демонстрирующую все разработанные элементы классов. Реализовать механизм наследования.

Вариант 16

Описать класс «поезд», содержащий следующие закрытые поля:

• название пункта назначения;

• номер поезда (может содержать буквы и цифры);

• время отправления.

Предусмотреть свойства для получения состояния объекта.

Описать класс «вокзал», содержащий закрытый массив поездов. Обеспечить следующие возможности:

• вывод информации о поезде по номеру с помощью индекса;

• вывод информации о поездах, отправляющихся после введенного с клавиатуры времени;

• перегруженную операцию сравнения, выполняющую сравнение времени отправления двух поездов;

• вывод информации о поездах, отправляющихся в заданный пункт назначения.

Информация должна быть отсортирована по времени отправления. Написать программу, демонстрирующую все разработанные элементы классов. Реализовать механизм наследования.

Вариант 17

Описать класс «товар», содержащий следующие закрытые поля:

• название товара;

• название магазина, в котором продается товар;

• стоимость товара в рублях.

Предусмотреть свойства для получения состояния объекта.

Описать класс «склад», содержащий закрытый массив товаров. Обеспечить следующие возможности:

• вывод информации о товаре по номеру с помощью индекса;

• вывод на экран информации о товаре, название которого введено с клавиатуры; если таких товаров нет, выдать соответствующее сообщение;

• сортировку товаров по названию магазина, по наименованию и по цене;

• перегруженную операцию сложения товаров, выполняющую сложение их цен.

Написать программу, демонстрирующую все разработанные элементы классов. Реализовать механизм наследования.

Вариант 18

Описать класс «самолет», содержащий следующие закрытые поля:

• название пункта назначения;

• шестизначный номер рейса;

• время отправления.

Предусмотреть свойства для получения состояния объекта.

Описать класс «аэропорт», содержащий закрытый массив самолетов. Обеспечить

следующие возможности:

• вывод информации о самолете по номеру рейса с помощью индекса;

• вывод информации о самолетах, отправляющихся в течение часа после введенного с клавиатуры времени;

• вывод информации о самолетах, отправляющихся в заданный пункт назначения;

• перегруженную операцию сравнения, выполняющую сравнение времени отправления двух самолетов.

Информация должна быть отсортирована по времени отправления. Написать программу, демонстрирующую все разработанные элементы классов. Реализовать механизм наследования.

Вариант 19

Описать класс «запись», содержащий следующие закрытые поля:

• фамилия, имя;

• номер телефона;

• дата рождения (массив из трех чисел).

Предусмотреть свойства для получения состояния объекта.

Описать класс «записная книжка», содержащий закрытый массив записей. Обеспечить следующие возможности:

• вывод на экран информации о человеке, номер телефона которого введен

с клавиатуры; если такого нет, выдать на дисплей соответствующее сообщение;

• поиск людей, день рождения которых сегодня или в заданный день;

• поиск людей, день рождения которых будет на следующей неделе;

• поиск людей, номер телефона которых начинается на три заданных цифры.

Написать программу, демонстрирующую все разработанные элементы классов. Реализовать механизм наследования.

Вариант 20

Описать класс «англо-русский словарь», обеспечивающий возможность хранения нескольких вариантов перевода для каждого слова. Реализовать доступ по строковому индексу — английскому слову. Обеспечить возможность вывода всех значений слов по заданному префиксу. Реализовать механизм наследования.

Вариант 21

Описать класс, обеспечивающий представление матрицы произвольного размера с возможностью изменения числа строк и столбцов, вывода на экран подматрицы любого размера и всей матрицы, доступа по индексам к элементу матрицы.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 22

Описать класс для работы с восьмеричным числом, хранящимся в виде строки символов. Реализовать конструкторы, свойства, методы и следующие операции:

• операции присваивания, реализующие значимую семантику;

• операции сравнения;

• преобразование в десятичное число;

• форматный вывод;

• доступ к заданной цифре числа по индексу.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 23

Описать класс «домашняя библиотека». Предусмотреть возможность работы с произвольным числом книг, поиска книги по какому-либо признаку (по автору, по году издания или категории), добавления книг в библиотеку, удаления книг из нее, доступа к книге по номеру.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 24

Написать класс «записная книжка». Предусмотреть возможность работы с произвольным числом записей, поиска записи по какому-либо признаку (например, по фамилии, дате рождения или номеру телефона),

добавления и удаления записей, сортировки по фамилии и доступа к записи по номеру.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 24

Написать класс «студенческая группа». Предусмотреть возможность работы с переменным числом студентов, поиска студента по какому-либо признаку (например, по фамилии, имени, дате рождения), добавления и удаления записей, сортировки по разным полям, доступа к записи по номеру.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 26

Написать класс, реализующий тип данных «вещественная матрица» и работу с ними. Класс должен реализовывать следующие операции над матрицами:

• сложение, вычитание (как с другой матрицей, так и с числом);

• комбинированные операции присваивания (+=, -=);

• операции сравнения на равенство/неравенство;

• операции вычисления обратной и транспонированной матрицы;

• доступ к элементу по индексам.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 27

Описать класс, реализующий тип данных «вещественная матрица» и работу с ними. Класс должен реализовывать следующие операции над матрицами:

• умножение, деление (как на другую матрицу, так и на число);

• комбинированные операции присваивания (*=, /=);

• операцию возведения в степень;

• методы вычисления детерминанта и нормы;

• доступ к элементу по индексам.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 28

Описать класс, реализующий тип данных «вещественная матрица» и работу с ними. Класс должен реализовывать следующие операции над матрицами:

• методы, реализующие проверку типа матрицы (квадратная, диагональная, нулевая, единичная, симметричная, верхняя треугольная, нижняя треугольная);

• операции сравнения на равенство/неравенство;

• доступ к элементу по индексам.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 29

Описать класс «множество», позволяющий выполнять основные операции: добавление и удаление элемента, пересечение, объединение и разность множеств.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Вариант 30

Описать класс «предметный указатель». Каждый компонент указателя содержит слово и номера страниц, на которых, это слово встречается. Количество номеров страниц, относящихся к одному слову, от одного до десяти. Предусмотреть возможность формирования указателя с клавиатуры и из файла, вывода указателя, вывода номеров страниц для заданного слова, удаления элемента из указателя.

Написать программу, демонстрирующую все разработанные элементы класса. Реализовать механизм наследования.

Шкала оценивания лабораторной работы

Примеры	1-5 балла
Примеры + Индивидуальное задание (без механизма наследования)	1-8 баллов
Примеры + Индивидуальное задание (с механизмом наследования)	1-10 баллов

Содержание отчета:

1. Номер и тема лабораторной работы.
2. Цель лабораторной работы.
3. Техническое оснащение.
4. Скриншоты выполнения примеров
5. При выполнении индивидуальных заданий в отчет внести изображение кода программы и окно выполнения программы.
6. Вывод по лабораторной работе