

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение
высшего образования «Казанский национальный исследовательский
технический университет им. А.Н. Туполева-КАИ»

Институт компьютерных технологий и защиты информации

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Лабораторный практикум

Казань 2021

Лабораторная работа №1

Структура консольного приложения.

Консольный ввод - вывод

Язык программирования C# является прямым наследником языка C++. Он унаследовал многие синтаксические конструкции языка C и объектно-ориентированную модель C++. В отличие от C++ C# является чисто объектно-ориентированным языком. В объектно-ориентированном программировании ход выполнения программы определяется объектами. Объекты это экземпляры класса. Класс это абстрактный тип данных, определяемый пользователем (программистом). Класс включает в себя данные и функции для обработки этих данных. В C# запрещены глобальные функции. Все функции должны быть обязательно определены внутри класса. Не является исключением и главная функция языка C# `Main()` (в отличии от языка C пишется с прописной буквы).

Объявление класса синтаксически имеет следующий вид:

```
class имя_класса
{
    // члены класса
}
```

Члены класса это данные и функции для работы с этими данными. Не затрагивая пока определения членов класса, рассмотрим общую структуру приложения для консоли. Для этого создадим пустой проект в интегрированной рабочей среде Microsoft Visual Studio.NET и проанализируем, что подготовит нам мастер приложения. Запустим на выполнение среду Microsoft Visual Studio.NET через кнопку «Пуск» панели задач. В открывшемся окне выберем пункт `File|New|Project`. В диалоговом

окне New Project необходимо выбрать тип проекта Visual C# Projects и шаблон приложения - Console Application. В поле наименование (Name) ввести новое имя проекта, либо согласиться с именем, которое предлагает мастер. В поле местоположение (Location) выбрать или ввести полный путь к рабочей папке проекта, например:

C:\work\4255

и затем щелкнуть мышкой на кнопке ОК.

При этом появится шаблон приложения, подготовленный для нас мастером:

```
using System;
namespace ConsoleApplication10
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Class1
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            //
            // TODO: Add code to start application here
            //
        }
    }
}
```

```
}  
}
```

Первая строка проекта *using System;*, включает в себя директиву *using*, которая сообщает компилятору, где он должен искать классы (типы), не определенные в данном пространстве имен. Мастер, по умолчанию, указывает стандартное пространство имен *System*, где определена большая часть типов среды .NET.

Следующей строкой *namespace ConsoleApplication10* мастер предложения определяет пространство имен для нашего приложения. По умолчанию в качестве имени выбирается имя проекта. Область действия пространства имен определяется блоком кода, заключенного между открывающей и закрывающей фигурными скобками. Пространство имен обеспечивает способ хранения одного набора имен отдельно от другого.

Имена, объявленные в одном пространстве имен не конфликтуют, при совпадении, с именами, объявленными в другом пространстве имен.

В шаблоне приложения имеется множество строк, которые являются комментариями.

В C# определены три вида комментариев:

- многострочный (*/*...*/*)
- однострочный (*//...*)
- XML (*///*) – комментарий для поддержки возможности создания самодокументированного кода.

Строка `[STAThread]` является атрибутом. Атрибуты задаются в квадратных скобках. С помощью атрибута в программу добавляется дополнительная описательная информация, связанная с элементом кода, непосредственно перед которым задается атрибут. В нашем случае указывается однопоточная модель выполнения функции `Main`.

Заголовок функции:

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

Функция Main определена как статическая (static) с типом возвращаемого значения void. Функция Main() C# как и функция main() языка C может принимать аргументы. Аргумент - это строковый массив, содержащий элементы командной строки. Тело функции пустое и в нем содержится, в виде комментария, предложение добавить туда код для запуска приложения:

```
// TODO: Add code to start application here
```

Воспользуемся этим предложением и добавим в тело функции одну строчку:

```
static void Main(string[] args)
{
    //
    // TODO: Add code to start application here
    Console.WriteLine("Привет!");
    //
}
```

затем, скомпилируем и запустим приложение на выполнение. Для этого необходимо выбрать пункт меню **Debug | Start Without Debugging** или нажать комбинацию клавиш **Ctrl+F5**. В результате выполнения программы появится окно со строкой приветствия.

Как вы, наверное, догадались, строка:

```
Console.WriteLine("Привет!");
```

выводит сообщение на консоль.

Функции консольного ввода-вывода являются методами класса Console библиотеки классов среды .NET.

Для ввода строки с клавиатуры используется метод `Console.ReadLine()`, а для ввода одного символа метод `Console.Read()`.

Для консольного вывода также имеются два метода:

- метод `Console.Write()`, который выводит параметр, указанный в качестве аргумента этой функции,
- метод `Console.WriteLine()`, который работает так же, как и `Console.Write()`, но добавляет символ новой строки в конец выходного текста.

Для анализа работы этих методов модифицируйте функцию `Main()` так, как показано ниже :

```
static void Main(string[] args)
{
    // TODO: Add code to start application here
    Console.WriteLine("Введите ваше имя");
    string str=Console.ReadLine();
    Console.WriteLine("Привет "+str+"!!!");
    Console.WriteLine("Введите один символ с
клавиатуры");
    int kod=Console.Read();
    char sim=(char)kod;
    Console.WriteLine("Код символа "+sim+" = "+kod);
}
```

Скомпилируйте и выполните приложение.

Проанализируйте код функции `Main()` и результат ее работы.

Здесь хорошо видно, что строку вывода метода `Console.WriteLine` можно формировать из переменных разного типа, просто объединяя их в одну строку с помощью знака `+`.

Можно сформировать точно такую же выходную строку, используя другую модификацию метода **Console.WriteLine**, которая принимает несколько параметров (список параметров), наподобие функции **printf()** языка программирования **Си**. Первым параметром списка является строка, содержащая маркеры в фигурных скобках. Маркер это номер параметра в списке. При выводе текста вместо маркеров будут подставлены соответствующие параметры из остального списка.

Для демонстрации работы этого метода вставьте в конец кода программы, строчку, как показано на листинге ниже.

```
static void Main(string[] args)
{
    // TODO: Add code to start application here
    Console.WriteLine("Введите ваше имя");
    string str=Console.ReadLine();
    Console.WriteLine("Привет "+str+"!!!");
    Console.WriteLine("Введите один символ с клавиатуры");
    int kod=Console.Read();
    char sim=(char)kod;
    Console.WriteLine("Код символа "+sim+" = "+kod);
    Console.WriteLine("Код символа {0} = {1}",sim,kod);
}
```

Скомпилируйте и выполните это приложение. Проанализируйте, полученный результат.

После маркера через запятую можно указать, сколько позиций отводится для вывода значений. Например, запись {1,3} означает, что для печати первого элемента списка отводится поле шириной в три символа. Причем, если значение ширины положительно, то производится выравнивание по правому краю поля, если отрицательно то по левому.

Добавив 4 новые строчки в конец кода функции Main(), убедимся в этом:

```
static void Main(string[] args)
{
    Console.WriteLine("Введите ваше имя");
    string str=Console.ReadLine();
    Console.WriteLine("Привет "+str+"!!!");
    Console.WriteLine("Введите один символ с клавиатуры");
    int kod=Console.Read();
    char sim=(char)kod;
    Console.WriteLine("Код символа "+sim+" = "+kod);
    Console.WriteLine("Код символа {0} – {1}",sim,kod);
    int s1=255;
    int s2=32;
    Console.WriteLine(" \n{0,5}\n+{1,4}\n-----\n{2,5}",s1,s2,s1+s2);
    Console.WriteLine(" \n{1,5}\n+{0,4}\n-----\n{2,5}",s1,s2,s1+s2);
}
```

Кроме того, после поля ширины через двоеточие можно указать форматную строку, состоящую из одного символа и необязательного значения точности.

Существует 8 различных форматов вывода:

C – формат национальной валюты,

D – десятичный формат,

E – научный (экспоненциальный) формат,

F – формат с фиксированной точкой,

G – общий формат,

N – числовой формат,

P – процентный формат,

X – шестнадцатеричный формат

Например, запись {2,9:C2} – означает, что для вывода второго элемента из списка, отводится поле шириной в 9 символов. Элемент выводится в формате денежной единицы с количеством знаков после запятой равной двум. При выводе результата происходит округление до заданной точности.

Для иллюстрации вышесказанного модифицируем функцию Main(), как показано ниже:

```
static void Main(string[] args)
{
    // TODO: Add code to start application here

    Console.WriteLine("Введите ваше имя");
    string str=Console.ReadLine();
    Console.WriteLine("Привет "+str+"!!!");
    Console.WriteLine("Введите один символ с клавиатуры");
    int kod=Console.Read();
    char sim=(char)kod;
    Console.WriteLine("Код символа "+sim+" = "+kod);
    Console.WriteLine("Код символа {0} = {1}",sim,kod);
    int s1=255;
    int s2=32;
    Console.WriteLine(" \n{0,5}\n+{1,4}\n-----\n{2,5}",s1,s2,s1+s2);
    Console.WriteLine(" \n{1,5}\n+{0,4}\n-----\n{2,5}",s1,s2,s1+s2);
    double sum1=500.3467;
    double sum2=43.5;
    Console.WriteLine(" \n{0,10:C2}\n+{1,9:C2}\n-----\n{2,10:C2}",
        sum1,sum2,sum1+sum2);
}
```

Скомпилируйте и запустите код на выполнение.

Проанализируйте изменения, происходящие с выходным текстом, при применении различных форматов.

Вопросы:

1. Какие статические методы класса **Console**, используются для ввода с клавиатуры?
2. Какой тип данных возвращает метод `Read()`?
3. Какой тип данных возвращает метод `ReadLine()`?
4. Какие статические методы класса `Console`, используются для вывода на консоль?
5. Как можно сформировать строку консольного вывода, используя знак сложения “+”?
6. Что означают числа, заключенные в `{...}`, форматной строки метода `Console.WriteLine("Код символа {0} = {1}",sim,kod);`?
7. Можно ли с помощью маркеров в `{...}` изменить порядок вывода параметров из списка метода `Console.WriteLine`, не изменяя порядка перечисления параметров в списке?
8. Как можно задать ширину поля вывода в символах?
9. Как можно организовать консольный вывод, выровненный по левому или по правому краю поля?
10. Как организовать вывод в шестнадцатеричном формате? Какие другие форматы вы знаете?
11. Как организовать вывод в формате с фиксированной точкой с точностью до трех знаков после запятой с округлением?

Задания:

Вывести на консоль значения функции в диапазоне углов от 0 до 180 градусов с шагом в 10 градусов. Для расчета использовать статические функции библиотечного класса Math. Вывод организовать в виде таблицы, состоящей из столбцов, выровненных по правому краю:

Первый столбец – значение аргумента функции в градусах, ширина поля вывода 4 символа.

Второй столбец – значение аргумента в радианах в формате с фиксированной точкой с точностью два знака после запятой с округлением, ширина поля вывода пять знаков.

Третий столбец – значение функции в формате с фиксированной точкой с точностью до четырех знаков после запятой, ширина поля вывода двенадцать знаков.

1. $F(x)=\sin(x)$;
2. $F(x)=\cos(x)$;
3. $F(x)=\sin(x)+\cos(x)$;
4. $F(x)=\operatorname{tg}(x)$;
5. $F(x)=\operatorname{ctg}(x)$;
6. $F(x)=\sinh(x)$;
7. $F(x)=\cosh(x)$;
8. $F(x)=\sinh(x)+\cosh(x)$;
9. $F(x)=\operatorname{tgh}(x)$;
10. $F(x)=\operatorname{ctgh}(x)$;

Лабораторная работа №2

Классы, член данные и член функции класса

Класс это абстрактный тип данных, определяемый программистом (пользователем).

С помощью классов определяются свойства объектов. Объекты это экземпляры класса.

Объявление класса синтаксически имеет следующий вид:

```
class имя_класса
{
    // члены класса
}
```

Члены класса – это данные и функции для работы с этими данными.

Имя класса – это, по сути дела, имя нового типа данных.

Создание экземпляра (объекта) класса осуществляется с помощью оператора new:

```
Имя_класса имя_объекта = new имя_класса();
```

Доступ к членам класса управляем. Управление доступом осуществляется с помощью спецификаций доступа:

- public – общедоступный член класса.
- private – член класса доступен только внутри данного класса.
- protected – член класса доступен только внутри данного класса и внутри классов, производных от данного.
- internal – член класса доступен только внутри данной сборки.

По умолчанию в классе устанавливается спецификация доступа `private`. Спецификация доступа, отличная от `private`, должна указываться явно перед каждым членом класса.

Данные класса подразделяются на *поля*, *константы* и *события*.

Поле – это обычная член-переменная, содержащая некоторое значение.

Можно, например, объявить класс, членами которого являются только поля:

```
class CA
{
    public      int      x;
    protected  float    z;
                double   m;
    public      char     sim;
    private    decimal   sum;
}
```

Обратите внимание, на то, что поле **m** здесь объявлено по умолчанию приватным.

Константы – это поле, объявленное с модификатором `const`, или, другими словами это поле, значение которого изменить нельзя, например:

```
public      const int x = 25;
```

События это члены класса, предназначенные для информирования клиентов класса, что, что-то произошло. В данной лабораторной работе события рассматриваться не будут.

Все не статические член функции класса имеют неограниченный доступ ко всем член данным класса независимо от спецификации доступа.

Методы

В основном, с помощью методов класса осуществляется обработка член данных класса. Другими словами, методы определяют поведение экземпляров данного класса. Методы класса это обычные функции C - стиля. В отличии от функций C, при передаче методу параметров по ссылке, необходимо указывать ключевое слово **ref** или **out**. Эти ключевые слова сообщают компилятору, что адреса параметров функции совпадают с адресами переменных, передаваемых в качестве параметров. Любое изменение значения параметров в этом случае приведет к изменению и переменных вызывающего кода. Рекомендуется для входного параметра использовать ключевое слово **ref**, а для выходного параметра ключевое слово **out**, так как параметр функции с ключевым словом **ref** должен быть обязательно проинициализирован перед вызовом функции. При вызове методов указание ключевых слов **ref** и **out** обязательно. Методы могут быть объявлены с ключевым словом **static** например:

```
public static int minabs(ref int x,ref int y)
{
    //тело функции
}
```

В этом случае для вызова метода имя класса, в котором она определена, и через точку имя метода:

```
Cmin.minabs(ref a,ref b);
```

Точка в C# означает принадлежность функции данному классу (в нашем случае Cmin).

Для закрепления изложенного материала создайте проект для консоли, введите текст приложения из примера 1.

Скомпилируйте и проанализируйте результаты работы приложения.

Пример 1:

```
using System;
namespace ConsoleApplication12
{
    class Cmin
    {
        public static int min(int x,int y)
        {
            int z = (x<y)?x:y;
            return z;
        }
        public static int minabs(ref int x,ref int y)
        {
            x = (x<0)?-x:x;
            y = (y<0)?-y:y;
            int z = (x<y)?x:y;
            return z;
        }
    }

    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            int a=-4;
```

```

        int b=2;
        Console.WriteLine("a={0} b={1}",a,b);
        int k =Cmin.min(a,b);
        Console.WriteLine("a={0} b={1}",a,b);
        Console.WriteLine("k="+k);
        k =Cmin.minabs(ref a,ref b);
        Console.WriteLine("a={0} b={1}",a,b);
        Console.WriteLine("k="+k);
    }
}
}

```

Свойства

Свойства в C# состоят из объявления поля и методов-аксессоров для работы с этим полем.

Эти методы- аксессоры называются получатель (get) и установщик (set).

Например, простейшее свойство **y**, работающее с полем **m**, можно представить следующим образом:

```

private int m=35;
public int y
{
    get
    {
        return m;
    }
    set
    {
        m=value;
    }
}

```

```
    }  
}
```

Свойство, определяется, так же как и поле, но после имени свойства идет блок кода, включающий в себя два метода `get` и `set`. Код этих методов может быть сколь угодно сложным, но в нашем случае это всего лишь один оператор. Аксессор **get** всегда возвращает значение того типа, который указан в определении свойства. Аксессор **set** всегда принимает в качестве параметра переменную **value**, которая передается ему неявно. Один из аксессоров может быть опущен, в этом случае мы получаем поле только для чтения или только для записи.

Обращение к свойству осуществляется точно так же как и к полю.

Пример 2:

```
using System;  
namespace ConsoleApplication11  
{  
    class CStatic  
    {  
        private int m=35;  
        public int y  
        {  
            get  
            {  
                return m;  
            }  
            set  
            {  
                m=value;  
            }  
        }  
    }  
}
```



```

    }
}
}
class Class1
{
    [STAThread]
    static void Main(string[] args)
    {
        // TODO: Add code to start application here
        CStatic p=new CStatic();//создается экземпляр класса
        Console.WriteLine("{0}",p.y);
        p.y=75;
        int z = p.y;
        Console.WriteLine("{0}",z);
    }
}
}

```

Введите текст программы из примера 2, скомпилируйте и проанализируйте работу приложения.

Индексаторы

Индексаторы позволяют приложению обращаться с объектом класса так, как будто он является массивом. Индексатор во многом напоминает свойство, но в отличие от свойства он принимает в качестве параметра индекс массива. Так как объект класса используется как массив, то в качестве имени класса используется ключевое слово **this**.

Пример 3:

```
using System;
namespace ConsoleApplication13
{
    class Rmas
    {
        protected float[] msf=new float[10];
        public float this[int j]
        {
            get
            {
                return msf[j];
            }
            set
            {
                msf[j]=value;
            }
        }
    }
    class Class1
    {
        static void Main(string[] args)
        {
            Rmas obj = new Rmas();
            for(int i=0; i<10;i++)
            {
                obj[i] = (float)1.5*i;
            }
            for(int i=0; i<10;i++)
            {
                Console.WriteLine("{0}",obj[i]);
            }
        }
    }
}
```

```
        }  
    }  
}
```

Вопросы:

1. Как объявляется класс?
2. Разрешается ли в объявлении структуры и класса инициализировать их член – данные?
3. Какие спецификации используются в классе и структуре для управления доступом к членам класса?
4. Чем отличаются по своему действию спецификации `private` от `protected`?
5. Какая спецификация доступа к данным устанавливается по умолчанию, при объявлении класса?
6. Какая спецификация доступа к данным устанавливается по умолчанию, при объявлении структуры?
7. Чем отличается передача параметров методам с помощью модификатора `ref` от передачи параметров методам с помощью модификатора `out`?
8. Чем отличается обращение к статическим методам класса от обращения к не статическим методам?
9. Что такое свойство?
10. Сколько свойств может быть объявлено в классе?
11. Может ли свойство использоваться для работы сразу с несколькими полями?
12. Сколько индексаторов может быть объявлено в классе для работы, например, с одномерными массивами?
13. Почему индексаторы называют интеллектуальными массивами?

Задания:

1. Введите текст программ из примеров 1,2.3, скомпилируйте и проанализируйте работу приложений.
2. Объявить класс, содержащий два поля целого типа, свойства для работы с этими полями, а также статическую функцию. Функция принимает два параметра целого типа и осуществляет обмен значений этих параметров. Создать объект этого класса. Ввести с клавиатуры два числа и присвоить значения этих чисел полям объекта. Распечатать поля объекта. Осуществить обмен значений полей объекта и вновь распечатать поля объекта.
3. Объявить класс, содержащий два поля целого типа, свойства для работы с этими полями, а также функцию. Функция возвращает сумму значений полей класса. Создать объект этого класса. Ввести с клавиатуры два числа и присвоить значения этих чисел полям объекта. Распечатать поля объекта и их сумму.
4. Объявить класс, содержащий одномерный целочисленный массив размерностью 5, индексатор для работы с этим массивом, а также функцию, для определения минимального и максимального элемента этого массива. Для хранения минимального и максимального значения предусмотреть два поля класса и свойства только для чтения этих полей. Создать объект этого класса. Ввести с клавиатуры элементы массива объекта. Распечатать элементы массива объекта, минимальное и максимальное значение массива.
5. Объявить класс, содержащий одномерный целочисленный массив размерностью 5, индексатор для работы с этим массивом, а также функцию, для определения суммы и средне - арифметического значения элементов этого массива. Для хранения суммы и среднего значения предусмотреть два поля класса и свойства только для чтения этих полей. Создать объект этого класса. Ввести с клавиатуры элементы массива объекта. Распечатать элементы массива объекта, суммы и средне - арифметического значение массива.

6. Объявить класс, содержащий одномерный целочисленный массив размерностью 5, индексатор для работы с этим массивом, а также функцию, для сортировки элементов этого массива в порядке возрастания. Класс также должен содержать поля для минимального и максимального элемента массива, а также свойства только для чтения этих полей. Значения этих полей определяются в результате работы функции сортировки. Создать объект этого класса. Ввести с клавиатуры элементы массива объекта. Распечатать элементы массива объекта, минимальное и максимальное значение массива.
7. Объявить класс, содержащий одномерный целочисленный массив размерностью 5, индексатор для работы с этим массивом, а также функцию, для сортировки элементов этого массива в порядке убывания. Класс также должен содержать поля для минимального и максимального элемента массива, а также свойства только для чтения этих полей. Значения этих полей определяются в результате работы функции сортировки. Создать объект этого класса. Ввести с клавиатуры элементы массива объекта. Распечатать элементы массива объекта, минимальное и максимальное значение массива.
8. Объявить класс, содержащий одномерный целочисленный массив размерностью 5, индексатор для работы с этим массивом. Создать три объекта данного класса. Ввести с клавиатуры элементы массива для первого и второго объекта. Элементы массива третьего объекта получить путем суммирования соответствующих элементов массивов первого и второго объектов. Распечатать массив третьего объекта.
9. Объявить класс, содержащий одномерный целочисленный массив размерностью 10, индексатор для работы с этим массивом. Создать объект данного класса. Ввести с клавиатуры элементы массива объекта. Найти элемент массива наиболее близкий по своему значению среднему арифметическому значению элементов массива. Распечатать значение этого элемента и его индекс.

10. Объявить класс, содержащий одномерный целочисленный массив размерностью 5, индексатор для работы с этим массивом. Создать три объекта данного класса. Ввести с клавиатуры элементы массива для первого и второго объекта. Элементы массива третьего объекта получить путем сравнения соответствующих элементов массивов первого и второго объектов и выбора наибольшего. Распечатать массив третьего объекта.

Лабораторная работа № 3

Конструкторы, поля только для чтения, вызов конструкторов

Конструктор – это метод класса, имеющий имя класса.

Конструкторов в классе может быть несколько или ни одного.

На конструкторы накладываются следующие ограничения:

1. Конструктор не может иметь возвращаемого значения даже **void**
2. Как следствие первого ограничения нельзя использовать оператор **return()**
3. Конструкторы нельзя объявлять виртуальными.

Конструктор автоматически вызывается на этапе компиляции при создании экземпляра данного класса. Попытка вызвать конструктор явным образом вызовет ошибку компиляции.

Различают следующие типы конструкторов:

1. Конструктор по умолчанию
2. Конструктор с аргументами

Конструктор по умолчанию

Конструктор, объявленный без аргументов, называется конструктором по умолчанию.

Если в классе не определен ни один конструктор, то компилятор сам автоматически создает конструктор по умолчанию, который инициализирует все поля класса своими значениями по умолчанию (для числовых значений ноль, для булевской переменной false, для строк пустые ссылки).

Пример 1:

// В классе СА нет явно объявленных конструкторов

```
using System;
```

```
namespace ConsoleApplication15
```

```
{
```

```
    class CA
```

```
    {
```

```
        public int x,y,z;
```

```
    }
```

```
class Class1
```

```
{
```

```
    [STAThread]
```

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

```
    {
```

```
        CA obj=new CA();
```

```
        Console.WriteLine("x={0,2} y={1,2} z={2,2}",obj.x,obj.y,obj.z);
```

```
    }
```

```
}
```

```
}
```

Добавим в определение класса CA конструктор по умолчанию и проследим, как изменится результат работы приложения:

```
class CA
```

```
{
```

```
    public int x,y,z;
```

```
    public CA()
```

```
    {
```

```
        x=3;
```

```
        y=2;
```

```
        z=x+y;
```

```
    }  
}
```

Конструктор с аргументами

Большинство конструкторов в C# принимают аргументы, с помощью которых разные объекты данного класса могут быть по разному инициализированы.

Пример 2:

```
using System;
```

```
namespace ConsoleApplication15
```

```
{
```

```
class CA
```

```
{
```

```
    public int x,y;
```

```
    public CA()
```

```
    {
```

```
        x=3;
```

```
        y=2;
```

```
    }
```

```
    public CA(int a)
```

```
    {
```

```
        x=a;
```

```
        y=2*a;
```

```
    }
```



```
public CA(int a,int b)
{
    x=a;
    y=b;
}
```

```
}
```

```
class Class1
```

```
{
```

```
[STAThread]
```

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

```
{
```

```
    CA obj=new CA();
```

```
    Console.WriteLine("x={0,2} y={1,2}",obj.x,obj.y);
```

```
    CA obj1=new CA(5);
```

```
    Console.WriteLine("x={0,2} y={1,2}",obj1.x,obj1.y);
```

```
    CA obj2=new CA(5,25);
```

```
    Console.WriteLine("x={0,2} y={1,2}",obj2.x,obj2.y);
```

```
}
```

```
}
```

```
}
```

Поля только для чтения

Поле только для чтения – это константное поле, значение которого изменить нельзя. Начальное значение поля только для чтения может быть вычислено в процессе выполнения приложения и установлено с помощью конструктора. Для объявления поля для чтения используется ключевое слово `readonly`.

Пример 3:

```
using System;
namespace ConsoleApplication15
{
    class CA
    {
        public int x;
        public readonly int rd;
        public CA()
        {
            x=3;
            rd=2*x;
        }
        public CA(int a)
        {
            x=a;
            rd=2*x;
        }
        public CA(int a,int b)
        {
```

```

        x=a;
        rd=b;
    }

}

class Class1
{
    [STAThread]
    static void Main(string[] args)
    {
        CA obj=new CA();
        Console.WriteLine("x={0,2}",obj.x);
        CA obj1=new CA(5);
        Console.WriteLine("x={0,2}",obj1.x);
        CA obj2=new CA(8,17);
        Console.WriteLine("x={0,2}",obj2.x);
        Console.WriteLine("поля для чтения{0,2} {1,2} {2,2}",
            obj.rd, obj1.rd, obj2.rd);
    }
}

```

Вызов конструкторов из других конструкторов.

Очень часто разные конструкторы независимо друг от друга содержат частично совпадающий код. С целью уменьшения объема программы рекомендуется, для выполнения совпадающего кода, осуществлять вызов конструкторов из других конструкторов. Для этого после имени

конструктора указывается, через двоеточие, ключевое слово `this` со списком передаваемых параметров:

```
public CA(int a,int b,int c):this(a,b)
{
    z=c;
}
```

При вызове конструкторов из других конструкторов, может быть вызван только один конструктор, который всегда выполняется первым.

Пример 4:

```
using System;
namespace ConsoleApplication15
{
    class CA
    {
        public int x,y,z;
        public readonly int rd;
        public CA()
        {
            x=3;
            y=2;
            z=x+y;
            rd=x+y+z;
        }
        public CA(int a,int b)
        {
            x=a;
            y=b;
```



```

        z=a+b;
        rd=x+y+z;
    }
    public CA(int a,int b,int c):this(a,b)
    {
        z=c;
        rd=x+y+z;
    }
}
class Class1
{
    [STAThread]
    static void Main(string[] args)
    {
        CA obj=new CA();
        Console.WriteLine("x={0,2} y={1,2} z={2,2}",obj.x,obj.y,obj.z);
        CA obj1=new CA(5,7);
        Console.WriteLine("x={0,2} y={1,2} z={2,2}",obj1.x,obj1.y,obj1.z);
        CA obj2=new CA(8,9,25);
        Console.WriteLine("x={0,2} y={1,2} z={2,2}",obj2.x,obj2.y,obj2.z);
        Console.WriteLine("поля для чтения  {0,2}  {1,2}
        {2,2}",obj.rd,obj1.rd,obj2.rd);
    }
}
}

```

Вопросы:

1. Как отличить конструктор класса от других функций класса?
2. Для чего используются конструкторы?
3. В каком случае компилятор сам создает конструктор по умолчанию?

4. Сколько в классе может быть объявлено не статических конструкторов по умолчанию?
5. Чем отличаются конструкторы с аргументами от конструкторов по умолчанию?
6. Как инициализируются поля только для чтения?
7. Чем отличаются константы от полей только для чтения?
8. Сколько конструкторов с аргументами может быть объявлено в классе?
9. Сколько конструкторов по умолчанию может быть объявлено в классе?
10. Как осуществляется вызов конструкторов данного класса из других конструкторов?
11. Допустимо ли объявлять конструкторы со спецификациями `protected` и `private`?

Задания:

1. Выполнить приложения из примеров 1-4. Проанализировать и объяснить работу приложений по примерам.
2. Создать приложение, в котором в цикле создается 10 объектов класса. Поля только для чтения каждого экземпляра равны порядковому номеру, отражающего очередность создания объектов.
3. Изменить тело конструктора с аргументами таким образом, чтобы производился анализ диапазона значения передаваемого аргумента. При выходе из некоторого диапазона - печаталось соответствующее сообщение, а значение аргумента приравнивалось, в зависимости от ситуации, минимально или максимально допустимому значению.
4. Объявить класс, содержащий два поля целого типа, свойства только для чтения этих полей, а также конструктор по умолчанию и конструктор с аргументами. Конструктор по умолчанию присваивает полям класса значение 2 и 3 соответственно. Конструктор с аргументами присваивает первому полю значение суммы аргументов,

а второму полю значение разности аргументов. Создать объекты этого класса с использованием всех конструкторов. Аргументы для конструкторов вводятся с клавиатуры. Распечатать поля объектов.

5. Объявить класс, содержащий поле целого типа две константы, задающие допустимый диапазон значения этого поля и свойства для работы с этими полями. Конструктор с двумя аргументами присваивает полю значение суммы аргументов, если сумма не выходит за пределы диапазона, в противном случае, значение суммы ограничивается в соответствии с допустимым верхним или нижним значением. Создать объект этого класса. Аргументы для конструктора вводятся с клавиатуры. Распечатать поля объекта.
6. Объявить класс, содержащий одномерный целочисленный массив размерностью 10, индексатор для работы с этим массивом, а также конструктор по умолчанию и конструктор с одним аргументом. Конструктор по умолчанию присваивает элементам массива значение равное индексу элемента. Конструктор с аргументами присваивает элементам массива значение равное сумме аргумента с индексом элемента. Создать объекты этого класса с использованием всех конструкторов. Аргументы для конструктора вводятся с клавиатуры. Распечатать поля (массивы) объектов.
7. Объявить класс, содержащий одномерный целочисленный массив, индексатор для работы с этим массивом, а также приватный конструктор, принимающий один параметр в качестве аргумента. Конструктор выделяет память под массив, размерность которого определяется аргументом конструктора. Также в классе определен конструктор с двумя аргументами, который при своей работе вызывает приватный конструктор, передавая ему свой первый аргумент в качестве параметра, а затем расписывает все элементы массива значением своего второго аргумента. Создать объект этого класса. Ввести с клавиатуры элементы массива объекта Аргументы для

конструктора вводятся с клавиатуры. Распечатать поля (массивы) объектов.

8. Объявить класс, содержащий одномерный целочисленный массив, а также функцию, для печати элементов этого массива. Конструктор класса принимает один параметр – ссылку на массив (внешний массив). Конструктор выделяет память под внутренний массив размерностью совпадающий с размерностью внешнего массива, затем значения элементов внешнего массива копирует во внутренний массив и сортирует внутренний массив по возрастанию. Класс также должен содержать поля `readonly` для минимального и максимального элемента массива. Значения этих полей определяются в результате работы конструктора. Создать объект этого класса. Элементы внешнего массива ввести с клавиатуры. Распечатать элементы массива объекта, минимальное и максимальное значение массива объекта.
9. Объявить класс, содержащий одномерный целочисленный массив, также функцию, для печати элементов этого массива. Конструктор класса принимает один параметр – ссылку на массив (внешний массив). Конструктор выделяет память под внутренний массив размерностью совпадающий с размерностью внешнего массива, затем значения элементов внешнего массива копирует во внутренний массив. Класс также должен содержать поля `readonly` для минимального и максимального элемента массива. Значения этих полей определяются в результате работы конструктора. Элементы внешнего массива ввести с клавиатуры. Создать объект этого класса . Распечатать элементы массива объекта, минимальное и максимальное значение массива объекта.
10. Объявить класс, содержащий одномерный целочисленный массив и функцию, для печати элементов этого массива. Конструктор класса принимает два параметра – ссылки на массивы (внешние массивы). Конструктор выделяет память под внутренний массив размерностью

совпадающий с размерностью большего внешнего массива, затем формируются значения элементов внутреннего массива путем попарного суммирования соответствующих элементов внешних массивов. Недостающие элементы одного из массивов заменяются при суммировании нулями. Класс также должен содержать поля readonly для минимального и максимального элемента массива. Значения этих полей определяются в результате работы конструктора. Элементы внешних массивов ввести с клавиатуры. Создать объект этого класса. Распечатать элементы массива объекта, минимальное и максимальное значение массива объекта.

11. Объявить класс, содержащий одномерный целочисленный массив и функцию, для печати элементов этого массива. Конструктор класса принимает два параметра – ссылки на массивы (внешние массивы). Конструктор выделяет память под внутренний массив размерностью совпадающий с размерностью большего внешнего массива, затем формируются значения элементов внутреннего массива путем попарного сравнения соответствующих элементов внешних массивов и записи во внутренний массив наименьшего значения. Недостающие элементы одного из массивов заменяются при сравнении нулями. Класс также должен содержать поля readonly для минимального и максимального элемента массива. Значения этих полей определяются в результате работы конструктора. Элементы внешних массивов ввести с клавиатуры. Создать объект этого класса. Распечатать элементы массива объекта, минимальное и максимальное значение массива объекта.
12. Объявить класс, содержащий одномерный целочисленный массив размерностью 10, индексатор для работы с этим массивом. Создать объект данного класса. Ввести с клавиатуры элементы массива объекта. Найти элемент массива наиболее близкий по своему значению

средне арифметическому значению элементов массива. Распечатать значение этого элемента и его индекс.

13. Объявить класс, содержащий одномерный целочисленный массив и функцию, для печати элементов этого массива. Конструктор класса принимает два параметра – ссылки на массивы (внешние массивы). Конструктор выделяет память под внутренний массив размерностью равной сумме размерностей внешних массивов, затем формируются значения элементов внутреннего массива путем копирования вначале элементов одного внешнего массивов, а затем второго. Результирующий массив сортируется. Класс также должен содержать поля readonly для минимального и максимального элемента массива. Значения этих полей определяются в результате работы конструктора. Элементы внешних массивов ввести с клавиатуры. Создать объект этого класса. Распечатать элементы массива объекта, минимальное и максимальное значение массива объекта

Лабораторная работа 4: Наследование

Цель работы:

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

Наследование

Наследование — это средство, с помощью которого один объект может приобретать свойства другого или другими словами это средство для получения новых типов данных из существующих типов. Класс, от которого осуществляется наследование, называется базовым классом или классом родителем, а класс наследник называют производным или порожденным классом. 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.

В C# для классов не поддерживается множественное наследование.

Синтаксис наследования:

```
class < производный класс > : < базовый класс >
{
.
.
.    //Члены производного класса
.
}
```

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’). Свойства базового класса при наследовании, как правило, расширяются.

Члены базового класса, объявленные со спецификацией **public** и **protected**, становятся членами производного класса с теми же спецификациями доступа, что и в базовом классе.

Пример:

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

namespace ConsolePrm7_1
{
    class CA
    {
        protected int x = 5;
        public int y = 25;
        protected int z;
        public CA()
        { z = 0; }
        public void Print()
        {
            Console.WriteLine("z = {0} x = {1} y = {2}", z, y, x);
        }
    }

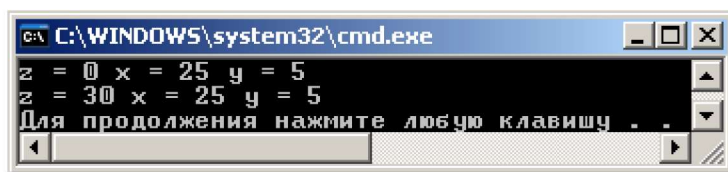
    class CB : CA    //Порождение нового класса
    {
        public CB()
        {
            z = x + y;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            CA ob = new CA();
            ob.Print();
            CB ob1 = new CB();
            ob1.Print();
        }
    }
}
```



```
}
```

Результат работы приложения:



Как видно по результатам работы приложения члены базового класса x, y, z , а также функция **Print**, становятся членами производного класса один к одному. А вот с закрытыми членами базового класса все не так однозначно.

Рассмотрим следующий пример.

Попытка скомпилировать следующий код:

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

namespace ConsolePrim7_2
{
    class CA
    {
        private int x = 25;
        public void PrintA()
        {
            Console.WriteLine("x = {0}", x);
        }
    }
    class CB : CA
    {
        public void PrintB()
        {
            Console.WriteLine("x = {0}", x); //Ошибка доступа
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        CA pA = new CA();
        CB pB = new CB();
        pA.PrintA();
    }
}
```

```

        pB.PrintB();
    }
}

```

вызовет ошибку компиляции:

Error 1 x is inaccessible due to its protection level

или в переводе на русский:

Ошибка 1 x недоступен из-за его уровня защиты.

Ошибку вызовет выделенная курсивом в тексте строка кода. На основании, того, что компилятор отказывается компилировать этот код, некоторые авторы делают вывод, что приватные члены класса не наследуются. Однако, попытка напечатать x с помощью открытой функции PrintA базового класса, которую производный класс CB получил в наследство, помощью кода приведенного ниже:

```

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

namespace ConsolePrim7_3
{
    class CA
    {
        private int x = 25;
        public void PrintA()
        {
            Console.WriteLine("x = {0} ",x);
        }
    }
    class CB : CA
    {
        public void PrintB()
        {
            Console.WriteLine("***CB***");
        }
    }

    class Program
    {

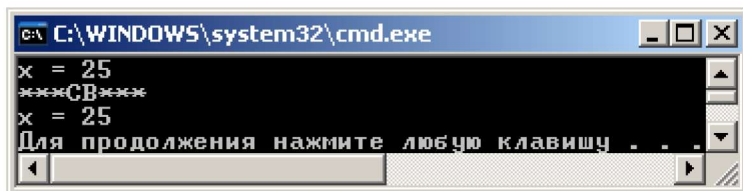
```

```

static void Main(string[] args)
{
    CA pA = new CA();
    CB pB = new CB();
    pA.PrintA();
    pB.PrintB();
    pB.PrintA();
}
}
}

```

Приводит к тому, что компиляция проходит без ошибок, и результат выполнения кода выглядит следующим образом:



Таким образом, можно сделать однозначный вывод, что приватные члены класса также наследуются, и к ним можно получить доступ через унаследованные открытые или защищенные функции базового класса.

Соккрытие методов базового класса

Если в производном классе объявить функцию с точно таким же заголовком, что и в базовом классе, то говорят, что новый метод сокрует метод базового класса.

Для метода, скрывающего метод базового класса, необходимо указать ключевое слово `new`, которое говорит, что вы сознательно пошли на этот шаг. Если не сделать этого, то компилятор выдает предупреждение, но ошибки не будет.

Например:

```

class CA
{
    string strA;

```

```

    public CA()
    {
        strA = "Привет";
    }

    public string func ( )
    {
        return strA;
    }
}

class CB:CA
{
    string strB;

    public CB()
    {
        strB = "HELLO";
    }

    public new string func ( )
    {
        return strB;
    }
}

class Class1
{
    [STAThread]
    static void Main(string[] args)
    {
        CA pA = new CA();
        CB pB = new CB();
        Console.WriteLine(pA.func()); // Привет
        Console.WriteLine(pB.func()); // HELLO
    }
}

```

}
}

Скомпилировать пример и проанализировать его работу.

Контрольные вопросы

1. Что понимается под термином «наследование»?
2. Какая классификация объектов соответствует наследованию?
3. Что общего имеет дочерний класс с родительским?
4. В чем состоит различие между дочерним и родительским классами?
5. Приведите синтаксис описания наследования классов в общем виде.

Проиллюстрируйте его фрагментом программы на языке C#.

6. Какому отношению соответствует иерархия классов?
7. Какому отношению соответствует иерархия объектов?
8. В чем заключается сокрытие функции базового класса?

Варианты заданий

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, двигатель внутреннего сгорания, дизель, реактивный двигатель

- 13) Республика, монархия, королевство, государство
- 14) Млекопитающее, парнокопытное, птица, животное
- 15) Корабль, пароход, парусник, корвет

Порядок выполнения работы:

- 1) Разработать методы и свойства для каждого из определяемых классов.
- 2) Реализовать программу на C# в соответствии с вариантом исполнения.
- 3) Подготовить отчет в твердой копии и в электронном виде.

Лабораторная работа 5

Полиморфизм

Цель работы:

Познакомиться с программированием полиморфных методов при объектно-ориентированном подходе при использовании языка C#.

Необходимые теоретические сведения

Свойства кода вести себя по-разному в зависимости от ситуации, возникающей в момент выполнения - называется *полиморфизмом*.

Различают два вида полиморфизма - специальный полиморфизм и чистый полиморфизм.

Специальный полиморфизм реализуется с помощью перегруженных функций.

Чистый полиморфизм - связан с механизмом наследования и реализуется через виртуальные функции..

В C# в отличие от C вызов некоторых функций на этапе трансляции только обозначается без точного указания какая именно функция вызывается. Какая из возможных функций будет вызвана, определяется на этапе выполнения программы. Такой процесс называется поздним связыванием.

Выбор во время выполнения соответствующей функции - члена среди функций основных и порожденных классов, в зависимости от объекта - называется чистым полиморфизмом. Чистый полиморфизм нельзя путать с перегрузкой функций, где аргументы функций различны по типу.

Перегруженная функция - член выбирается во время компиляции, т.е. для нее осуществляется раннее связывание.

Возможность динамически выбирать функцию-член, в зависимости от объекта, осуществляется с помощью виртуальных функций.

Виртуальные методы

Виртуальные функции объявляются с помощью ключевого слова **virtual**. Это слово при объявлении функции определяет для нее механизм позднего связывания. Виртуальными могут быть только не статические функции-члены,

так как характеристика `virtual` наследуется.

Виртуальная функция является обычным выполняемым кодом. В порожденном классе она может быть переопределена, то есть для нее может быть написано новое тело.

Переопределение виртуальных функций

Функция, объявленная в производном классе, переопределяет виртуальную функцию в базовом классе только тогда, когда имеет то же имя и работает с тем же количеством и типом аргументов, что и виртуальные функции базового класса. Если они отличаются хоть одним аргументом, то функция в производном классе считается совершенно другой (новой) и переопределения не происходит.

Это трудноуловимая ошибка, когда программист, считая, что он переопределяет виртуальную функцию базового класса, на самом деле, ошибившись в одном каком либо знаке заголовка функции – определяет новую, в C++ никак не “отлавливается” транслятором. В C#, для предотвращения подобной ошибки, в отличие от C++, необходимо, явно указывать с помощью ключевого слова `override`, что переопределяется функция базового класса. Встретив ключевое слово `override`, транслятор проверяет, имеется ли функция с таким заголовком среди виртуальных функций базового класса. В случае не совпадения заголовков выводится сообщение об ошибке компиляции.

Если функция объявлена `virtual`, то это свойство передается всем переопределениям в порожденных классах.

Переопределять виртуальные функции в порожденном классе необязательно. В этом случае функция базового класса становится функцией производного класса (по наследству).

Можно с помощью ключевого слова `sealed` запретить переопределение функции базового класса в производном классе. В этом случае попытка переопределения приведет к ошибке трансляции.

Правила вызова методов (функций)

Если метод не виртуальный, то вызывается метод типа (класса), на который указывала ссылка в момент ее объявления.

Если метод является виртуальным, то при выполнении кода будет проверяться, куда на самом деле указывает ссылка (на какой именно объект). Затем проверяется экземпляром, какого класса является этот объект и вызывается соответствующий метод этого класса.

Пример:

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

namespace ConsolePrim8
{
    class CA
    {
        string nameClass;
        public CA()
        {
            nameClass = "Class CA ";
        }
        public void Fnc()
        {
            Console.WriteLine(nameClass + "Fnc()");
        }

        public virtual void Fvrt()
        {
            Console.WriteLine(nameClass + "Fvrt()");
        }
    }

    class CB : CA
    {
        string nameClass;
        public CB()
        {
```



```

        nameClass = "Class CB ";
    }
    public new void Fnc()
    {
        Console.WriteLine(nameClass + "Fnc()");
    }

    public override void Fvrt()
    {
        Console.WriteLine(nameClass + "Fvrt()");
    }
}

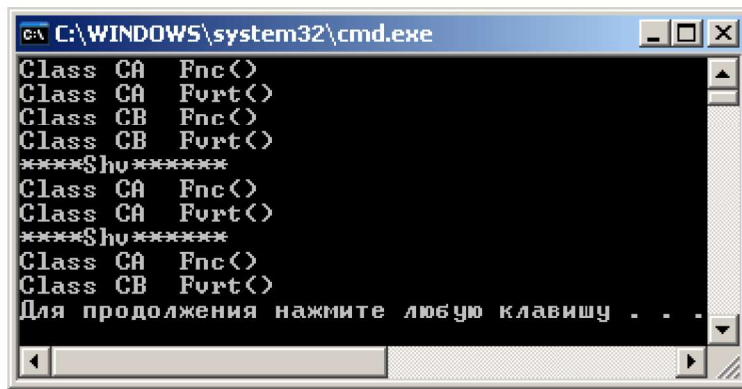
class Program
{
    static void Shw(CA m)
    {
        Console.WriteLine("****Shv*****");
        m.Fnc(); //Раннее связывание
        m.Fvrt(); //Позднее связывание
    }

    static void Main(string[] args)
    {
        CA a = new CA();
        CB b = new CB();
        a.Fnc(); // CA-Fnc()
        a.Fvrt(); // CA-Fvrt()
        b.Fnc(); // CB-Fnc()
        b.Fvrt(); // CB-Fvrt()
        Shw(a); // CA-Fnc()
                // CA-Fvrt()
        Shw(b); // CA-Fnc()
                // CB-Fvrt()

    }
}

```

Результат работы приложения:



В приведенном примере в классе родителе и производном классе содержатся функции с одинаковыми именами. Одна из этих функций Fvrt() является виртуальной и переопределяется в производном классе. Поэтому, в функции Shw , транслятор должен при вызове функции Fvrt() использовать механизм позднего связывания. Какая именно функция Fvrt() класса CA или CB будет вызвана на этапе выполнения приложения, зависит от объекта, который передается функции Shw() с помощью ссылки в качестве параметра. Вторую не виртуальную функцию Fnc() транслятор жестко свяжет еще на этапе компиляции с классом CA, так как функция Shw() принимает в качестве параметра ссылку на класс CA. Рассмотрим еще один пример, который несколько проще по коду, чем предыдущий. Здесь, класс родитель CA имеет в своем составе всего одну виртуальную функцию Display.

При вызове этой функции выводится сообщение: "Функция CA". В производном классе CB эта функция переопределяется таким образом, что при вызове этой функции выводится сообщение: "Функция CB".

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

namespace ConsolePrm8_1
{
    class CA
    {
        public virtual void Display()
        {
            Console.WriteLine("Функция CA");
        }
    }
}
```

```

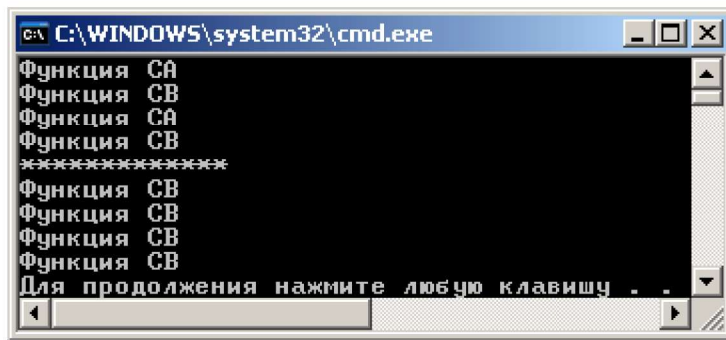
    }
class CB:CA
{
    public override void Display()
    {
        Console.WriteLine("Функция CB");
    }
}
class Program
{
    static void Show(CA m)
    {
        m.Display();
    }
    static void Main(string[] args)
    {
        CA pA = new CA();
        CB pB = new CB();
        pA.Display();
        pB.Display();
        Show(pA);
        Show(pB);
        pA = pB;
        Console.WriteLine("*****");
        pA.Display();
        pB.Display();
        Show(pA);
        Show(pB);

    }
}
}

```

В функции Main нашего приложения создается по одному объекту данных классов, последовательно вызываются функция Display каждого объекта, а также, ссылки на эти объекты последовательно передаются функции Show. Затем, значение одной ссылки присваивается значению другой ссылки (pA = pB;)и последовательность вызовов функций Display и Show вновь повторяется.

Результат работы приложения:



наглядно демонстрирует как изменяется вывод при выполнении одной и той же последовательности операторов.

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. Например:

```
abstract void func();
```

Использование `virtual` совместно с `abstract` недопустимо – приведет к возникновению ошибки на этапе трансляции.

Модификатор `abstract` не применим к `static`.

Класс, содержащий один или более методов `abstract`, также должен быть объявлен как абстрактный:

Пример:

```
abstract class abstr
{
    int x;
```



```
public abstract void func();  
:  
}
```

Абстрактные классы создаются для нужд производных классов.

Поскольку абстрактный класс не определен полностью, объекты этого класса создать невозможно.

При объявлении класса, производного от абстрактного класса, все абстрактные методы должны быть переопределены.

Таким образом, абстрактный класс определяет интерфейсы некоторых методов, но не их реализацию.

Контрольные вопросы

- 1) Что понимается под термином «полиморфизм»?
- 2) В чем состоит основной принцип полиморфизма?
- 3) В чем состоит значение основного принципа полиморфизма?
- 4) Какие механизмы используются в языке C# для реализации концепции полиморфизма?
- 5) Что понимается под термином «виртуальный метод»?
- 6) Какое ключевое слово языка C# используется для определения виртуального метода?
- 7) В чем состоит особенность виртуальных методов в производных (дочерних) классах?
- 8) В какой момент трансляции программы осуществляется выбор версии виртуального метода?
- 9) Какие условия определяют выбор версии виртуального метода?
- 10) Какое ключевое слово (модификатор) языка C# используется для определения виртуального метода в базовом (родительском) классе?
- 11) Какое ключевое слово (модификатор) языка C# используется для определения виртуального метода в производном (дочернем) классе?
- 12) Какие модификаторы недопустимы для определения виртуальных методов?

- 13) Что означает термин «переопределенный метод»?
- 14) В какой момент трансляции программы осуществляется выбор вызываемого переопределенного метода?
- 15) Приведите синтаксис виртуального метода в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.
- 16) Что понимается под термином «абстрактный класс»?
- 17) В чем заключаются особенности абстрактных классов?
- 18) Какой модификатор языка C# используется при объявлении абстрактных методов?
- 19) Являются ли абстрактные методы виртуальными?
- 20) Используется ли модификатор `virtual` языка C# при объявлении абстрактных методов?
- 21) Возможно ли создание иерархии классов посредством абстрактного класса?
- 22) Возможно ли создание объектов абстрактного класса?
- 23) Приведите синтаксис абстрактного класса в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.

Варианты заданий

Расширить иерархию классов из лабораторной работы №4 с использованием виртуального класса в качестве основы иерархии. Показать пример использования полиморфизма методов.

Порядок выполнения работы:

- 1) Изменить иерархию классов и реализовать ее на C#.
- 2) Показать на примере одного из методов, присутствующих в каждом классе, свойство полиморфизма.
- 3) Подготовить отчет в твердой копии и в электронном виде.

Лабораторная работа №6

Делегаты и события

По своей структуре делегат - это объект, который ссылается на метод. С помощью делегата можно вызвать метод, на который он указывает.

В процессе выполнения программы делегату можно присвоить ссылку на другой метод. Это дает возможность определять во время выполнения программы, какой из методов должен быть вызван.

Делегаты реализуются как экземпляры классов, производных от библиотечного класса `System.Delegate`. Для создания делегата необходимо выполнить два шага.

На первом шаге необходимо объявить делегат. При этом сигнатура делегата должна полностью соответствовать сигнатуре метода, который он представляет.

Например, делегат должен ссылаться на статический метод класса `CA`:

```
static int min(int x,int y),
```

тогда объявление делегата может выглядеть, следующим образом:

```
delegate int LpFunc(int a,int b);
```

На втором шаге мы должны создать экземпляр делегата для хранения сведения о представляемом им методе:

```
LpFunc pfnk = new LpFunc(CA.min);
```

Экземпляр делегата может ссылаться на любой статический метод или метод объекта любого класса, при условии, что сигнатура метода полностью соответствует сигнатуре делегата.

Пример 1:

```
using System;
namespace ConsoleApplication14
{
    class MathOprt
    {
        public static double Mul2(double val)
        {
            return val*2;
        }
        public static double Sqr(double val)
        {
            return val*val;
        }
    }
    delegate double Dblop(double x); //объявление делегата
    class Class1
    {
        static void Main(string[] args)
        {
            Dblop [] operation = // создание экземпляров делегата
            {
                new Dblop(MathOprt.Mul2),
                new Dblop(MathOprt.Sqr)
            };
            for(int j=0;j<operation.Length;j++)
            {
                Console.WriteLine("Результаты операции[ {0} ]:",j);
                Prc(operation[j], 4.0);
            }
        }
    }
}
```

```

        Prc(operation[j], 9.94);
        Prc(operation[j], 3.143);
    }
}

static void Prc(DblOp act, double val)
{
    double rslt = act(val);
    Console.WriteLine("Исходное значение {0}, результат {1}",
        val,rslt);
}
}
}

```

Делегаты могут хранить несколько адресов областей памяти. То есть делегат может указывать на несколько различных методов. Это позволяет, последовательно инициализируя адреса вызывать метод за методом. Эта способность делегатов называется *многоадресностью делегатов*.

Для создания цепочки вызовов методов необходимо сначала создать экземпляр делегата для одного метода, а затем с помощью операции “ + = ” добавить остальные методы. В процессе выполнения кода можно не только добавлять новые методы, но и удалять не нужные с помощью операции ”- =”.

Методы, представляемые многоадресными делегатами должны возвращать значение **void**.

Перепишем код из примера 1 с применением многоадресного делегата.

Пример 2:

```

using System;
namespace ConsoleApplication14
{
    class MathOprt

```



```

{
public static void Mul2(double val)
{
    double rslt= val*2;
    Console.WriteLine("Mul2 исходное значение {0},результат {1}",
        val,rslt);
}
public static void Sqr(double val)
{
    double rslt = val*val;
    Console.WriteLine("Sqr исходное значение {0}, результат {1}",
        val,rslt);
}
}
delegate void DbOp(double x);//объявление делегата

class Class1
{
    [STAThread]
    static void Main(string[] args)
    {
        DbOp operations = new DbOp(MathOprt.Mul2);
        operations += new DbOp(MathOprt.Sqr);
        Prc(operations, 4.0);
        Prc(operations, 9.94);
        Prc(operations, 3.143);
    }
    static void Prc(DbOp act, double val)
    {
        Console.WriteLine("\n*****\n");
    }
}

```

```

        act(val);
    }
}

```

События

Работа с событиями осуществляется в C# согласно модели «издатель-подписчик». Класс, ответственный за инициализацию (выработку) событий публикует событие, и любые классы могут подписаться на это событие. При возникновении события исполняющая среда уведомляет всех подписчиков о произошедшем событии, при этом вызываются соответствующие методы-обработчики событий подписчиков. Какой обработчик события будет вызван – определяется делегатом.

Платформа .NET требует для всех обработчиков событий следующей сигнатуры кода:

```

void OnRecChange(object source, EventArgs e)
{
    // Код для обработки события
}

```

Обработчики событий обязательно имеют тип возвращаемого значения **void**. Обработчики событий принимают два параметра. Первый параметр `_` это ссылка на объект, сгенерировавший событие. Эта ссылка передается обработчику самим генератором событий. Второй параметр – это ссылка на объект класса `EventArgs` или класса производного от него. В производном классе может содержаться дополнительная информация о событии.

Обработчик события определяется делегатом. Согласно сигнатуре обработчика события делегат должен принимать два параметра и выглядеть следующим образом:

```
public delegate void ChangeEventHandle(object source, ChangeEventArgs e);
```

Для того, чтобы иметь возможность подписаться на событие класс генератора событий должен содержать член типа указанного делегата с ключевым словом **event** и метод, который будет вызываться при возникновении события, например:

```
public event ChangeEventHandler OnChangeHandler;
```

Этот член является специализированной формой многообъектного делегата. Используя операцию “+=”, клиенты могут подписаться на это сообщение:

```
gnEvent.OnChangeHandler +=  
    new GenEvent.ChangeEventHandler (OnRecChange);
```

где gnEvent имя класса генератора событий.

Нижеследующий пример демонстрирует работу с событиями.

Пример№3:

```
using System;  
namespace sobit  
{  
    class ChangeEventArgs : EventArgs  
    {  
        string str;  
        public string Str  
        {  
            get  
            {
```

```

        return str;
    }
}
int change;
public int Change
{
    get
    {
        return change;
    }
}
public ChangeEventArgs(string str,int change)
{
    this.str = str;
    this.change = change;
}
}

class GenEvent    // Генератор событий - издатель
{
    public delegate void ChangeEventHandler
        (object source,ChangeEventArgs e);

    public event ChangeEventHandler OnChangeHandler;

    public void UpdateEvent(string str,int change)
    {
        if(change==0)
            return;

        ChangeEventArgs e =
            new ChangeEventArgs(str,change);
    }
}

```



```

        if (OnChangeHandler != null)
            OnChangeHandler(this,e);
    }

}

//Подписчик
class RecEvent
{
    //Обработчик события
    void OnRecChange(object source,ChangeEventArgs e)
    {
        int change = e.Change;
        Console.WriteLine("Вес груза '{0}' был {1} на {2} тонны",
            e.Str,change > 0 ? "увеличен" : "уменьшен",
            Math.Abs(e.Change));
    }

    // в конструкторе класса осуществляется подписка
    public RecEvent(GenEvent gnEvent)
    {
        gnEvent.OnChangeHandler += //здесь осуществляется подписка
            new GenEvent.ChangeEventHandler(OnRecChange);
    }
}

class Class1
{
    [STAThread]
    static void Main(string[] args)
    {

```

```

        GenEvent gnEvent = new GenEvent();
        RecEvent inventoryWatch = new RecEvent(gnEvent);
        gnEvent.UpdateEvent("грузовика", -2);
        gnEvent.UpdateEvent("автопоезда", 4);
    }
}
}

```

Вопросы:

1. Можно ли по сигнатуре объявления делегата определить сигнатуру функции, которую представляет делегат.
2. Какие ограничения накладываются на функции, которые может представлять многоадресный делегат?
3. Как включить или исключить заданную функцию из списка функций, представляемых многоадресным делегатом?
4. Как объявляется событие?
5. Что такое событие?
6. Чем отличается событие от многоадресного делегата?
7. Какова общепринятая сигнатура обработчика события?
8. Как осуществляется генерация события?
9. Каким образом обработчику события передается дополнительная информация о произошедшем событии?
10. Как осуществляется «подписка» на событие?

Задания:

1. Создать приложение, в котором генератор события “снабжает” событие следующей информацией: название поезда, время прибытия, номер вагона и места. Приемник события распечатывает эту информацию.
2. Создать приложение, в котором генератор события “снабжает”

событие следующей информацией: название поезда, станция назначения, станция отправления и время в пути. Приемник события распечатывает эту информацию.

3. Создать приложение, в котором генератор события после генерации первого события генерирует последующие события только в том случае, если приемник события уведомляет, что сообщение принято (квитирование). Для квитирования использовать первый параметр обработчика события.
4. Создать приложение, в котором генератор события после генерации первого события генерирует только определенное количество событий. Количество генераций определяется путем уведомления со стороны приемника. Для уведомления использовать первый параметр обработчика события.
5. Создать приложение, в котором генератор события после генерации первого события генерирует последующие события только в том случае, если приемник события уведомляет, что событие принято (квитирование). Для квитирования использовать второй параметр обработчика события.
6. Создать приложение, в котором генератор события после генерации первого события генерирует только определенное количество событий. Количество генераций определяется путем уведомления со стороны приемника. Для уведомления использовать второй параметр обработчика события.
7. Создать приложение, в котором генератор события может генерировать три разных события. Приемники событий выступают в качестве абонентов почтового отделения и могут пересылать друг другу информацию, используя генератор в качестве почтового ящика. При этом они указывают номер (от 1 до 3) следующего приемника и некоторое целое число, которое передается получателю. Такой цикл передачи продолжается до тех пор, пока

какой либо из приемников в качестве получателя не укажет номер ноль. В этом случае приложение завершает свою работу. При запуске приложения первое почтовое извещение всегда получает от генератора первый приемник. Для адресации и передачи информации использовать второй аргумент обработчика события.

8. Выполнить задание по пункту 7, но адресации и передачи информации использовать первый аргумент обработчика события.
9. Создать приложение, в котором генератор события, путем генерации одного события запрашивает у трех приемников некоторый ресурс. Каждый приемник сообщает, какое количество ресурса он может выделить. Для передачи информации использовать второй аргумент обработчика события.
10. Выполнить задание по пункту 9, но для передачи информации использовать первый аргумент обработчика события.

Лабораторная работа № 7

Windows – приложение

Любое окно приложения для Windows , представляет собой форму, порожденную от класса System.Windows.Forms.Form.

Откроем новый проект – проект Windows Application. Проект создается по аналогии с Console Application. Перед нами появится пустая поверхность окна приложения – форма. Размер, местоположение, цвет фона, имя заголовка и другие свойства формы можно изменять с помощью окна свойств – Properties (отображение свойств выбранного элемента и событий, связанных с ним).

Панель Properties содержит следующие разделы свойств:

Accessibility	– доступность
Appearance	– появление
Behavior	– поведение
Configurations	– конфигурации
Data	– данные
Design	– проект
Focus	– фокус
Layout	– размещение
Window style	– стиль окна

Используя раздел свойств Appearance можно изменять:

- цвет поверхности – BackColor (по умолчанию такого же цвета будут все элементы управления);
- фоновую картинку – BackgroundImage,
- вид курсора – Cursor,
- цвет шрифта – ForeColor,
- стиль рамки окна – FormBorderStyle,
- текст заголовка формы – Text ,

- местоположение текста заголовка формы – RightToLeft

Используя раздел свойств Layout можно изменять:

- размер окна – Size
- параметр, определяющий начальную позицию – StartPosition,
- координаты левого верхнего угла формы – Location (актуален, если StartPosition = Manual),
- вид начального отображения формы (минимизированное, нормальное или максимизированное) – WindowState,
- размер формы в максимизированном состоянии – MaximizeBox,
- размер формы в минимизированном состоянии – MinimizeBox,

Используя раздел свойств Window style можно изменять:

- вид иконки – Icon
 - доступность кнопки максимизации – MaximizeBox
 - доступность кнопок минимизации – MinimizeBox
- и другие.

Легко заметить, что мы можем изменять размеры формы не только с помощью панели свойства (Properties), но и с помощью мыши, при этом размеры (Size) будут меняться автоматически.

Поэкспериментируйте с этими и другими свойствами, посмотрите изменения на форме, связанные с изменением свойств.

Так же можно увидеть панели:

- Server Explorer (подключение к проекту БД),
- Solution Explorer (отображение подключенных проектов, файлов, пространств имен и т.п.),
- Dynamic Help (отображение справочной информации),

- Toolbox (на поверхность формы можно добавлять различные элементы управления)

и другие (в зависимости от настроек пользователя этих панелей можно не обнаружить!).

Возможности, связанные с другими панелями, так же весьма интересны, но их рассмотрение не входит в цели данной лабораторной работы.

Добавить дополнительные панели можно через пункт меню «View».

Теперь, когда вы более или менее усвоили свойства (Properties), связанные с формой, можно рассмотреть события. На той же панели свойств есть кнопка с пиктограммой «молния». Щелкнем на ней. При этом отобразятся различные события, связанные:

- с мышью (движение, движение в определенных направлениях, вход или выход из определенной области, нажатиями клавиши мыши),
- с нажатием клавиши клавиатуры,
- с рисованием,
- с изменением стиля,
- с размером окна,

и многие другие (подробнее с теми или иными событиями можно ознакомиться самостоятельно, используя источники с более глубокой детализацией материала).

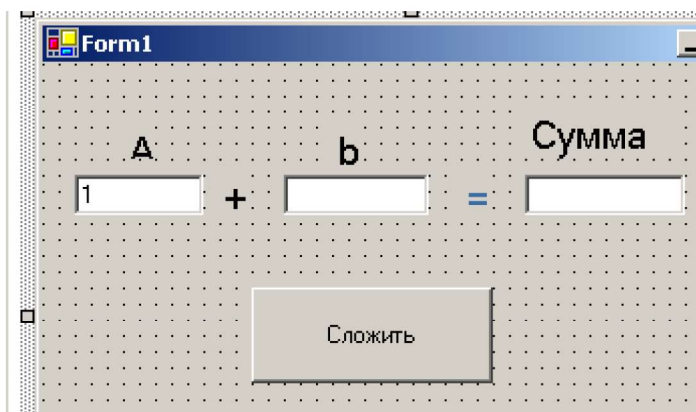
Чтобы задействовать то или иное событие достаточно лишь дважды щелкнуть на его названии и компилятор автоматически создаст имя и тело метода, который будет обрабатывать выбранной событие. Так же можно ввести самостоятельно имя метода и сменить фокус от выбранного события – тело метода создастся автоматически с введенным именем! Существует еще и третий способ: сначала создается метод обрабатывающий событие, а затем выбирается в выпадающем списке! В тело метода необходимо поместить все, что необходимо для обработки события.

Чтобы показать, как все это работает, создадим простенькое приложение

– калькулятор для сложения двух чисел.

Для этого, во-первых, удаляем все «набросанные» на поверхность формы элементы управления, поверхность – должна быть пустой.

Во-вторых, «выкладываем» (перетаскиваем) из Toolbox на форму два



Textbox, пять Label и одну Button.

Изменив свойства Text Label на – «А», «В», «+», «=» и «Сумма» добиваемся, чтобы это выглядело примерно как на рисунке.

Затем, для удобства изменим имена (Name, а не Text!)

Textbox и Button. Первому Textbox присваиваем имя «А», второму – «В», третьему – «С», а Button – «Calculate». Все это делается, если Вы еще не забыли, в панели свойств (не забывайте при этом выделить нужный элемент!). Теперь нам нужно добавить обработку лишь одного события – нажатие кнопки. Это можно сделать всеми перечисленными выше способами или просто двойным щелчком по кнопке. И сразу же автоматически появляется код приложения с телом метода, обрабатывающего событие нажатия кнопки.

Пояснение. При двойном щелчке по любому элементу управления автоматически создается метод по обработке того или иного события, но(!!!) создается всегда, то событие обработка, которого более характерна для данного элемента управления. Для кнопки – нажатие на кнопку, для формы – загрузка и т.п.

В теле метода нужно дописать код, который будет выглядеть следующим образом:

```
private void Calculate_Click(object sender, System.EventArgs e)
{
    double a, b;
```



```

try/*Обработка исключений, здесь выделяется блок кода, в котором
могут возникнуть исключения (ошибки)*/
{
    a=Convert.ToDouble(A.Text);/*Считываем текст (A.Text),
        затем конвертируем его в формат double*/
    b=Convert.ToDouble(B.Text);/*Считываем текст (B.Text),
        затем конвертируем его в формат double*/
    C.Text=Convert.ToString(a+b);/*Присваиваем тексту третьего
редактора (C.Text) конвертированную в строковый формат сумму
чисел(a+b)полученных из двух редакторов*/
}
catch//Обработка исключений – произошла ошибка
{
    MessageBox.Show("Проверьте правильность ввода чисел!");
}
}

```

Скомпилируем приложение. Если все-таки что-то не работает, проверьте, все ли вы сделали правильно! Свертись с полным листингом приложения.

Добавим в наше приложение событие нажатия кнопки мыши.

В тело метода обработки этого события можно необходимо вставить следующий код:

```

private void Form1_MouseDown(object sender,
System.Windows.Forms.MouseEventArgs e)
{//создается объект типа Color для хранения цвета
Color clr=Color.FromArgb((int)200,(int)0,(int)0);
Pen p=new Pen(clr);//Создаем цветное перо
Graphics g=CreateGraphics();//Создаем контекст графического устройства

```

```

g.DrawRectangle(p,30,180,100,20);//Рисуем прямоугольник
p=System.Drawing.Pens.Fuchsia;//Выбираем перо цвета фуксин
g.DrawLine(p,30,254,68,275);//Рисуем прямую линию
/*Здесь рисуем шарик окружностями
увеличивая его радиус и меняя цвет каждой окружности*/
float R2,r2,k=1,
R=120;//максимальный радиус шара
R2=R*R;
int x,y;
int red=20,gre=250,blu=150;
for(y=0;y<=R;y++)
    for(x=0;x<=y;x++)
    {
        r2=(float)2*x*x;
        if(r2>R2) break;
        k=1-r2/R2;
        clr=Color.FromArgb((int)(k*red),(int)(k*gre),(int)(k*blu));
        p=new Pen(clr);
        /* Именно здесь происходит вывод окружностей на
поверхность*/
        g.DrawEllipse(p,(float)(210-0.5*x),(float)(200-0.5*x),(float)x,(float)x);
    }
}

```

Скомпилировать и выполнить приложение.

Вопросы:

1. Какие разделы свойств содержит панель Properties.
2. В теле какой функции происходит создание элементов управления Windows?
3. Что означает свойство TabIndex элемента управления?
4. Как задается местоположение и размер элементов управления с

помощью мыши?

5. Как задается местоположение и размер элементов управления с помощью панели Properties?
6. Как создаются обработчики событий с помощью панель Properties?
7. Каким образом можно задать имя обработчика события, отличное от имени, предлагаемого мастером приложения?
8. В теле какой функции мастер создания приложения осуществляет “подписку” на событие?
9. Обработчик, какого события формируется при двойном щелчке на поверхности формы?
10. Обработчик, какого события формируется при двойном щелчке на поверхности элемента управления?

Задания:

1. Создать приложение, в котором, при щелчке мыши на поверхности кнопки увеличивалась прозрачность формы в 2 раза.
2. Создать приложение, в котором, при щелчке мыши на поверхности кнопки текст заголовка формы изменялся на текст окна редактора текста (TextBox).
3. Создать приложение, в котором, при щелчке мыши на поверхности кнопки текст метки Label изменялся на текст окна редактора текста (TextBox).
4. Создать приложение, в котором, при щелчке мыши на поверхности кнопки текст на поверхности кнопки изменялся на текст окна редактора текста (TextBox).
5. Создать приложение, в котором, при щелчке мыши на поверхности кнопки высота кнопки в пикселях изменялась на

число в окне редактора текста (TextBox).

6. Создать приложение, в котором, при щелчке мыши на поверхности кнопки высота формы в пикселях изменялась на число в окне редактора текста (TextBox).
7. Создать приложение, в котором, при щелчке мыши на поверхности кнопки ширина кнопки в пикселях изменялась на число в окне редактора текста (TextBox).
8. Создать приложение, в котором, при щелчке мыши на поверхности кнопки ширина формы в пикселях изменялась на число в окне редактора текста (TextBox).
9. Создать приложение, в котором, при щелчке мыши на поверхности одной кнопки цвет фона формы становится, синим, а при нажатии на другую кнопку, красным.
10. Создать приложение, в котором, при щелчке мыши на поверхности кнопки местоположение кнопки на поверхности формы изменялось в соответствии с координатами, введенными в окна редакторов текста (TextBox).
11. Создать приложение, в котором, при щелчке мыши на поверхности кнопки местоположение формы на поверхности экрана монитора изменялось в соответствии с координатами, введенными в окна редакторов текста (TextBox).

Лабораторная работа № 8

Простейшие графические возможности

В пространстве имен System.Drawing – определены основные структуры для представления:

- точки (координат) – Point и PointF
- размера – Size и SizeF
- прямоугольных областей – Rectangle и RectangleF.

Буква F в конце названия структуры означает, что, в отличие от обычных структур (без F), поля структуры имеют не целочисленные значения, а значения вещественного типа (float).

Структура Size

Структура Size предназначена для хранения ширины и высоты объекта и имеет, для этого, соответствующие открытые свойства Width и Height, доступные как для записи, так и для чтения. При создании объекта Size, с помощью конструктора по умолчанию:

```
Size sz = new Size();
```

свойства Width и Height устанавливаются в ноль.

Изменить значения свойств в последствии можно, например, следующим образом:

```
sz.Width = 40;  
sz.Height = 60;
```

Структура содержит два конструктора с аргументами:

```
public Size(int, int);  
public Size(Point);
```

Конструкторы с аргументами позволяют, в момент создания, инициализировать разные экземпляры структуры по-разному:

```
Size sz1= new Size(10,20);    // sz1.Width = 10, sz1.Height = 20  
Size sz2 = new Size(15,50);   // sz2.Width = 15, sz2.Height = 50
```

Структура Point

Структура Point содержит открытые свойства X и Y целого типа, доступные, как для записи, так и для чтения.

Для создания точки "pt" можно использовать конструктор по умолчанию:

```
Point pt = new Point();
```

Конструктор по умолчанию при создании точки обнуляет значения свойств X и Y.

В дальнейшем координаты точки можно изменить, например, следующим образом:

```
pt.X=25;  
pt.Y=75;
```

Инициализировать новую точку класса Point, можно используя, конструкторы с аргументами:

```
public Point(Size);  
public Point(int, int);
```

Например:

```
Point pt1 = new Point(10,20); // pt1.X=10, pt1.Y=20  
Size sz = new Size(27,45);  
Point pt2 = new Point(sz);    // pt2.X=27, pt2.Y=45
```

Открытый метод структуры:

```
public void Offset( int dx, int dy);
```

изменяет текущие координаты точки по формулам:

$X=X+dx$ и $Y=Y+dy$;

Структура Rectangle

Структура предназначена для определения координат и размера прямоугольника. Структура содержит открытые свойства, часть из которых доступна только для чтения, а часть, как для чтения, так и для записи.

В структуре определены два конструктора с аргументами:

```
public Rectangle(  
    int x,           // x-координата левого верхнего угла прямоугольника  
    int y,           // y-координата левого верхнего угла прямоугольника  
    int width,       // ширина прямоугольника  
    int height       // высота  прямоугольника  
);
```

```
public Rectangle(  
    Point location,  // координата левого верхнего угла прямоугольника  
    Size size        // размер  прямоугольника  
);
```

С помощью этих конструкторов можно определять размеры и местоположение прямоугольников при их создании:

```
Point pt = new Point(10,15);  
Size sz = new Size (50,70);  
Rectangle rct = new Rectangle(pt,sz);  
Rectangle rect = new Rectangle(20,20,50,30);
```

Структура Rectangle содержит ряд доступных методов. Рассмотрим некоторые из них.

Метод:

```
public void Intersect(Rectangle);
```

Возвращает структуру, которая описывает прямоугольник, представляющий пересечение двух прямоугольников. Если не имеется никакого пересечения, все свойства структуры обнуляются.

Например:

```
Rectangle rect,rct;  
rect = new Rectangle(20,25,50,55);  
rct = new Rectangle(10,10,30,40);  
rect.Intersect(rct);
```

выполнение, приведенного фрагмента кода установит значения свойства структуры прямоугольника rect следующим образом:

X=20, Y=25, Width=20, Height=25.

Метод:

```
public static Rectangle Union( Rectangle a, Rectangle b);
```

Возвращает структуру, описывающий минимальный по размерам

прямоугольник, включающий в себя прямоугольники a и b.

Методы `public void Offset(Point pos)` и `public void Offset(int x, int y)` смещают координаты левой верхней точки прямоугольника по обеим осям на величину, задаваемую параметрами методов.

Представление цвета

Представление цвета осуществляется с помощью экземпляров структуры `System.Drawing.Color`.

Для задания цвета используется статический метод класса:

```
public static Color.FromArgb( int red, int green, int blue);
```

Параметры метода `red`, `green` и `blue` задают интенсивность красной, зеленой и синей составляющей цвета. Значение каждой компоненты цвета может изменяться в диапазоне от 0 до 255. Это позволяет отобразить 2^{24} различных цветов.

Для задания цвета можно также использовать один из перегруженных методов `FromArgb`:

```
public static Color FromArgb(int alpha, Color cr);  
public static Color FromArgb(int alpha, int red, int green, int blue);
```

Дополнительный параметр `alpha` задает степень прозрачности цвета. Чем меньше это число, тем меньше насыщенность цвета и тем более прозрачным является этот цвет. Значение параметра `alpha` может изменяться в диапазоне от 0 до 255. Значение 0 определяет полностью прозрачный (бесцветный), а значение 255 – полностью насыщенный (непрозрачный) цвет.

Структура `Color` содержит более 200 статических свойств, доступных

только для чтения. Эти свойства задают именованные или, так называемые, Интернет – цвета, которые правильно воспроизводятся всеми WEB браузерами. Например:

```
Color clr2 = Color.Beige;      // бежевый
Color clr3 = Color.Magenta;    // сиреневый
Color clr4 = Color.Orange;     // оранжевый
```

Кисти и перья

Графические объекты рисуются с помощью перьев и кистей.

Сплошные кисти создаются как экземпляры класса System.Drawing.SolidBrush, например:

```
Brush br2 = new SolidBrush(Color.Magenta);
Brush br3 = new SolidBrush(Color.FromArgb(200,10,120));
```

Параметр color конструктора public SolidBrush(Color color) класса SolidBrush задает цвет сплошной кисти.

В классе System.Drawing.Brushes определено большое количество статических свойств, возвращающих кисть Интернет цветов. Создание таких кистей выглядит следующим образом:

```
Brush brr = Brushes.Orange;
```

В классе System.Drawing.Drawing2D.HatchBrush определены штриховые кисти.

Конструкторы класса:

```
public HatchBrush(HatchStyle hatchstyle, Color foreColor, Color backColor);
public HatchBrush(HatchStyle hatchstyle, Color foreColor);
```

Параметры:

foreColor – цвет штриха кисти;

backColor – цвет фонового штриха кисти (по умолчанию – черный цвет);

hatchstyle – стиль штриховой кисти.

Существует большое количество доступных стилей, наиболее типичными являются:

Cross – решетчатая кисть;

DiagonalCross – диагональная решетчатая кисть;

Horizontal – горизонтальная штриховка;

Vertical – вертикальная штриховка.

Например, создание кисти с вертикальной штриховкой синего цвета и фоновым штрихом бежевого цвета будет выглядеть следующим образом:

```
Brush br1 = new HatchBrush(HatchStyle.Vertical,Color.Blue,Color.Beige);
```

Перья описываются классом System.Drawing.Pen.

Конструкторы класса:

```
public Pen(Color color);
```

```
public Pen(Color color, float width);
```

```
public Pen( Brush brush);
```

```
public Pen(Brush brush, float width);
```

Параметры:

color – цвет пера;
width – толщина пера;
brush – кисть.

Примеры создания перьев:

```
Pen pn = new Pen(Color.Magenta);  
Pen pn1 = new Pen(Color.Orange,5);  
Pen pn2 = new Pen(Brushes.Orange);  
Pen pn3 = new Pen(Brushes.Magenta,10);  
Pen pn4 = new Pen(Color.FromArgb(125,155, 0));  
Pen pn5 = new Pen(Color.FromArgb(25,155,200),10);
```

В классе `System.Drawing.Pens` содержится множество статических свойств, описывающих перья с интернет цветом и толщиной в один пиксель. Создание таких перьев выглядит следующим образом:

```
Pen pn6 = Pens.Brown;  
Pen pn7 = Pens.Magenta;
```

Рисование линий и фигур

Для вывода текстовой и графической информации в окно приложения необходимо использовать контекст устройства.

Контекст устройства в среде .NET инкапсулирован («завернут») в базовом классе `System.Drawing.Graphics`. Для создания объекта класса `Graphics` необходимо использовать метод `CreateGraphics()`, возвращающий ссылку на

объект класса Graphics:

```
Graphics dc = CreateGraphics();
```

Полученный объект dc содержит контекст устройства, позволяющий осуществлять вывод информации в окно приложения.

Класс Graphics содержит множество методов, позволяющих рисовать различные графические фигуры. Рассмотрим некоторые из них.

Рисование контуров прямоугольников осуществляется с помощью методов:

```
public void DrawRectangle( Pen pen, Rectangle rect);  
public void DrawRectangle( Pen pen, int x, int y, int width, int height);  
public void DrawRectangle( Pen pen, float x, float y, float width, float height);
```

Рисование контуров эллипсов осуществляется с помощью методов:

```
public void DrawEllipse ( Pen pen, Rectangle rect);  
public void DrawEllipse ( Pen pen, int x, int y, int width, int height);  
public void DrawEllipse ( Pen pen, float x, float y, float width, float height);
```

Рисование закрашенных эллипсов и прямоугольников осуществляется с помощью методов:

```
public void FillEllipse( Brush brush, Rectangle rect);  
public void FillEllipse( Brush brush, int x, int y, int width, int height);  
public void FillEllipse( Brush brush, float x, float y, float width, float height);  
public void FillRectangle( Brush brush, Rectangle rect);  
public void FillRectangle( Brush brush, int x, int y, int width, int height);  
public void FillRectangle( Brush brush, float x, float y, float width, float height);
```

Параметры методов означают следующее:

pen – перо;

brush – кисть;

rect – прямоугольник;

x – координата x левого верхнего угла прямоугольника;

y – координата y левого верхнего угла прямоугольника;

width – ширина прямоугольника;

height – высота прямоугольника;

Рисование линий осуществляется с помощью перегруженных методов:

```
public void DrawLine(Pen pen, Point pt1, Point pt2);
```

```
public void DrawLine(Pen pen, PointF pt1, PointF pt2);
```

```
public void DrawLine(Pen pen, int x1, int y1, int x2, int y2);
```

```
public void DrawLine(Pen pen, float x1, float y1, float x2, float y2);
```

Параметры методов означают следующее:

pen – перо;

pt1 – начальная точка рисования;

pt2 – конечная точка рисования;

x1 и y1 – координаты начальной точки рисования;

x2 и y2 – координаты конечной точки рисования;

Примеры использоания функций:

```
dc.DrawRectangle(Pens.OrangeRed,5,10,25,45);
```

```
dc.DrawEllipse(Pens.Magenta,100,125,20,15);
```

```
dc.FillEllipse(Brushes.BlueViolet,45,50,20,15);
```

```
dc.DrawLine(Pens.Green,20,40,60,70);
```

Рисование текста

Для рисования текста используют перегруженный метод DrawString.

Рассмотрим два из шести перегруженных методов DrawString:

```
public void DrawString(string s, Font fnt, Brush br, PointF pt);
```

```
public void DrawString(string s, Font fnt, Brush br, RectangleF ltRct);
```

Параметры:

s – строка символов,

fnt – шрифт текста,

br – кисть,

pt – точка, определяющая координаты вывода текста,

ltRct – прямоугольник, внутри которого выводится текст, если же текст не вмещается в область прямоугольника, то он (текст) обрезается.

Например:

```
Font fnt = new Font("Arial",10); //Шрифт Arial, размер 10
```

```
dc.DrawString("Привет!",fnt, Brushes.Green,10,20);
```

Пример 1. Создадим приложение, которое при щелчке левой кнопкой мыши на окне приложения выводит в месте щелчка прямоугольник с текстом координатами левого верхнего угла прямоугольника, а при щелчке правой кнопкой мыши в месте щелчка выводится закрашенный эллипс.

Для этого создайте проект Windows Application. В окне свойств формы выбрать событие MouseDown, дважды щелкнуть на названии события мышкой. В появившейся заготовке метода-обработчика события вставить код, чтобы тело метода выглядело следующим образом:

```
private void Form1_MouseDown(object sender,
```



```

System.Windows.Forms.MouseEventArgs e)
{
    Graphics dc = CreateGraphics();
    Font fnt = new Font("Coyrier",10);
    if(e.Button.ToString()=="Left")
    {
        dc.DrawRectangle(Pens.OrangeRed,e.X,e.Y,15,15);
        dc.DrawString("X="+e.X.ToString()+"
Y="+e.Y.ToString(),fnt,Brushes.Green,e.X,e.Y+20);
    }
    if(e.Button.ToString()=="Right")
    {
        dc.DrawEllipse(Pens.Magenta,e.X,e.Y,20,15);
        dc.FillEllipse(Brushes.Blue,e.X,e.Y,20,15);
    }
}

```

Скомпилируйте приложение. Проанализируйте полученные результаты.

Перерисовка окна приложения

Если свернуть окно приложения из предыдущего примера, затем вновь развернуть его, то мы, к сожалению, заметим, что изображение на поверхности окна исчезло. Операционная система не восстанавливает содержимого окна. Восстановлением графики и текста должно заниматься само приложение. Операционная система в необходимых случаях вырабатывает сообщение (событие Paint), которое «говорит», что окно приложения не корректно и его необходимо перерисовать. Перерисовка окна должна происходить по событию Paint. Метод-обработчик этого события имеет заголовок:

```
private void Form_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
```


Для этого метода нет необходимости создавать контекст устройства, он передается методу с помощью параметра `e`. Для получения контекста устройства необходимо выполнить следующую операцию:

```
Graphics dc = e.Graphics;
```

В теле этой функции необходимо выполнить все действия для перерисовки окна.

Очень часто перерисовка окна должна происходить в определенные моменты времени по инициативе приложения. Это бывает необходимо при выводе на экран анимации.

«Заставить» операционную систему выработать событие Paint можно путем вызова метода `Invalidate()`, который является членом класса `System.Windows.Forms.Form`. Существуют несколько перегруженных версий этого метода. Одна из них принимает в качестве параметра прямоугольник, который определяет область окна для перерисовки. Используемая нами версия без параметров перерисовывает все окно.

Пример 2.

Создадим приложение, в котором при запуске появляется прямоугольник, ширина которого увеличивается через каждые 150мс на 5 пикселей.

1. Объявим в классе `Form1` переменную:

```
private int xWidth;
```

2. Создадим метод-обработчик события Paint, для этого в свойствах формы выберем событие Paint и щелкнем по нему. После редактирования тела метода, метод должен иметь вид:

```
private void Form1_Paint(object sender,
    System.Windows.Forms.PaintEventArgs e)
{
    Graphics dc = e.Graphics;
    dc.DrawRectangle(Pens.RoyalBlue,10,100,xWidth,50);
}
```

3. Перенесем на форму Timer из Toolbox, затем в свойствах таймера установим свойство Enable в true и Interval в 150.

4. Далее двойным щелчком мыши по таймеру создаем метод-обработчик таймера. После редактирования тела метода, метод должен иметь вид:

```
private void timer1_Tick(object sender, System.EventArgs e)
{
    xWidth += 5;
    Invalidate();
}
```

Скомпилируйте приложение. Проанализируйте полученные результаты.

Вопросы:

1. Какой метод используется для «смещения» координат точки, прямоугольника?
2. Как определить находится ли точка или прямоугольник внутри другого прямоугольника?
3. С помощью какого метода можно найти пересечение прямоугольников?
4. С помощью какого метода можно найти минимальный по размеру прямоугольник, включающий в себя два прямоугольника?
5. С помощью экземпляров, какой структуры задается цвет?
6. Как создаются сплошные кисти, штриховые кисти?
7. Как создаются сплошные перья, штриховые перья?
8. Что означает Интернет цвет, Интернет кисть, Интернет перо?

9. Что собой представляет интерфейс графических устройств?
10. Что такое контекст устройства? Как он создается?
11. С помощью, каких методов контекста устройств осуществляется рисование линий?
12. С помощью, каких методов контекста устройств осуществляется рисование прямоугольников?
13. С помощью, каких методов контекста устройств осуществляется рисование эллипсов?
14. Как задать тип и размер шрифта текста?
15. Как можно нарисовать текст по заданным координатам, в заданном прямоугольнике?
16. Какая функция, обработчик события Paint, вызывается для перерисовки окна приложения?
17. С помощью вызова, какой функции можно инициализировать ОС для генерации события Paint?

Задание:

1. Создайте приложение, в котором по нажатию клавиши мыши в месте щелчка выводилось название нажатой клавиши мыши.
2. Создайте приложение, в котором при щелчке на правую кнопку мыши рисовалась линия, соединяющая координату щелчка с левым верхним углом окна приложения.
3. Создайте приложение, в котором выводился след движения мыши (использовать функцию рисования линии).
4. Создать приложение, в котором вводятся координаты окружности и меняются по нажатию кнопки.
5. Создать приложение, в котором прямоугольник смещается на пять пикселей при каждом щелчке мышью вне прямоугольника в ту сторону, с которой был произведен щелчок.
6. Создать приложение, в котором прямоугольник увеличивается на пять

пикселей при каждом щелчке правой кнопкой мыши и уменьшается при каждом щелчке левой кнопкой мыши на поверхности прямоугольника.

7. Создать приложение, в котором появляется маленький прямоугольник, который увеличивается (эффект наезда).
8. Создать приложение, в котором в момент создания формы появляются два прямоугольника равных размеров и координат, затем прямоугольники начинают разъезжаться по разным углам окна (по диагонали).
9. Нарисовать минимальный по размеру прямоугольник, включающий в себя оба прямоугольника по пункту 8.
10. Нарисовать прямоугольник, являющийся пересечением исходных прямоугольников по пункту 8.

Лабораторная работа №9

Создание меню

Меню, находящееся между заголовком и клиентской областью окна приложения, называется *главным меню*. Кроме главного меню существует другой вид меню *контекстные меню* или их еще называют *меню быстрого вызова*. Пункты в видимой части главного меню называются пунктами *верхнего уровня*. Выбор любого пункта верхнего уровня главного меню приводит к появлению в виде прямоугольника подменю или другими словами дочернего меню. Подменю в свою очередь состоит из нескольких пунктов. В общем случае меню верхнего уровня представляет собой массив пунктов меню, а каждый пункт меню – массив пунктов подменю.

Для создания меню используются классы *MainMenu*, *ContextMenu* и *MenuItem*, порожденные от абстрактного класса *Menu*. В классе *Menu* объявлен внутренний класс *MenuItemCollection*, который наследуется классами *MainMenu*, *ContextMenu* и *MenuItem*.

Для создания главного меню можно использовать один из трех конструкторов класса *MainMenu*:

```
public MainMenu()  
public MainMenu(MenuItem[] menuItems)  
public MainMenu.IContainer container)
```

После создания объекта *MainMenu*, необходимо вызвать свойство формы *Menu* и передать ему имя объекта *MainMenu*. Свойство *Menu* формы доступно как для записи, так и для чтения. Это свойство позволяет заменять в процессе работы приложения, одно меню формы другим.

Для создания контекстного меню можно использовать один из двух

конструкторов класса `ContextMenu`:

```
public ContextMenu ()  
public ContextMenu (MenuItem[] menuItems)
```

Обычно контекстное меню связано с, каким либо, элементом управления, то есть при щелчке правой кнопкой мыши на поверхности элемента управления появляется контекстное меню. Для того чтобы связать объект, являющийся контекстным меню, с элементом управления необходимо использовать свойство `ContextMenu` элемента, унаследованное им от класса `Control`. Свойство `ContextMenu` доступно как для записи, так и для чтения.

Контекстное меню не обязательно связывать с конкретным контрольным элементом. Его можно вызвать в любом обработчике события, используя метод:

```
public void Show (Control control, Point pos),
```

где первый параметр – это ссылка на элемент управления, к которому относится контекстное меню, а второй – координата точки, где будет выведено контекстное меню.

Для создания конечных пунктов меню можно использовать один из конструкторов класса `MenuItem`:

```
public MenuItem (),  
public MenuItem (string text),  
public MenuItem (string text, EventHandler onClick),  
public MenuItem (string text, MenuItem[] items),  
public MenuItem (string text, EventHandler onClick, Shortcut shortcut),
```

где:

`text` – название пункта меню,

`onClick` – объект типа `EventHandler`, который является делегатом. С помощью

`onClick` вызывается функция обработчик пункта меню.

`shortcut` – комбинация клавиш для быстрого вызова пункта меню.

Shortcut - это перечисление допустимых клавиатурных сочетаний для меню. Оно содержит около 150 различных сочетаний. Использовать в качестве комбинации клавиш, не определенных в перечислении не допустимо.

В качестве примера создадим приложение, в котором имеется главное меню из двух пунктов и каждый пункт содержит два подпункта. При выборе любого пункта подменю на экран монитора выводится окно сообщения MessageBox текст, которого соответствует названию пункта. Подменю первого пункта меню верхнего уровня может вызываться с помощью быстрых клавиш ALT+1 и ALT+2.

Пример 1:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Menu1
{
    public partial class Form1 : Form
    {
        // Объявление ссылок
        MainMenu MnMen1; //Главное меню
        //Первый пункт меню верхнего уровня
        MenuItem pnkt1;
        MenuItem pnkt1_1; //Первый пункт подменю
        MenuItem pnkt1_2; //Второй пункт подменю
        //Второй пункт меню верхнего уровня
    }
}
```



```
MenuItem pnkt2;  
MenuItem pnkt2_1; //Первый пункт подменю  
MenuItem pnkt2_2; //Второй пункт подменю
```

```
public Form1()      //Конструктор формы  
{  
    InitializeComponent();  
  
    this.Text = "МЕНЮ"; //Заголовок формы  
  
    // Создаем первый пункт меню верхнего уровня - массив из подпунктов  
    pnkt1_1 = new MenuItem("Подпункт 1_1", new EventHandler(Msg1_1),  
        Shortcut.Alt1);  
    pnkt1_2 = new MenuItem("Подпункт 1_2", new EventHandler(Msg1_2),  
        Shortcut.Alt2);  
    pnkt1 = new MenuItem("Пункт 1", new MenuItem[] { pnkt1_1, pnkt1_2 });  
  
    // Создаем второй пункт меню верхнего уровня - массив из подпунктов  
    pnkt2_1 = new MenuItem("Подпункт 2_1", new EventHandler(Msg2_1));  
    pnkt2_2 = new MenuItem("Подпункт 2_2", new EventHandler(Msg2_2));  
    pnkt2 = new MenuItem("Пункт 2", new MenuItem[] { pnkt2_1, pnkt2_2 });  
  
    // Создаем главное меню - массив из пунктов верхнего уровня  
    MnMen1 = new MainMenu(new MenuItem[] { pnkt1, pnkt2 });  
  
    this.Menu = MnMen1; //Связываем меню с формой  
}
```

//Ниже представлены обработчики пунктов меню


```

void Msg1_1(object sr, EventArgs e)
{
    MessageBox.Show("Подпункт 1_1");
}
void Msg1_2(object sr, EventArgs e)
{
    MessageBox.Show("Подпункт 1_2");
}
void Msg2_1(object sr, EventArgs e)
{
    MessageBox.Show("Подпункт 2_1");
}
void Msg2_2(object sr, EventArgs e)
{
    MessageBox.Show("Подпункт 2_2");
}
}
}

```

Скомпилируйте и выполните приложение.

Обратите внимание, что сначала создаются пункты подменю, затем из этих пунктов создается пункт меню верхнего уровня, и в последнюю очередь из пунктов меню верхнего уровня “собирается” главное меню.

Если необходимо в подменю отделить визуально одну группу пунктов от другой горизонтальной линией, то необходимо объявить пункт меню следующим образом:

```
MenuItem pnkt = new MenuItem("-");
```

Прочерк в качестве имени подменю распознается как разделитель.

Класс `MenuItem` содержит большое количество свойств. Используя эти свойства можно динамически изменять внешний вид меню, управлять

доступом к пунктам меню и т.п.

Некоторые из них:

1. `Shortcut` - позволяет устанавливать новое сочетание клавиш для быстрого доступа к меню,
2. `ShowShortcut` - установка значения свойства `false` запрещает вывод, справа от названия пункта меню, сочетания клавиш быстрого доступа к меню, `true` - разрешает
3. `Text` - позволяет изменять название пунктов меню
4. `Visible` – разрешает (`true`) - запрещает (`false`) отображение пунктов меню.
5. `Enabled` – блокирует пункт меню при установке его в значение `false` и разблокирует при установке в `true`. Заблокированный пункт меню становится серым.
6. `DefaultItem` – установка его в `true` приводит к тому, что данный пункт меню становится пунктом меню по умолчанию, то есть, при двойном щелчке на пункте меню верхнего уровня, это пункт подменю сразу вызывается. Текст заголовка такого подменю выводится полужирным шрифтом.
7. `Checked` – установка его в `true` приводит к тому, что слева от названия пункта подменю выводится галочка.

В классе `Menu` объявлен внутренний класс `MenuItemCollection`, который наследуется классами `MainMenu`, `ContextMenu` и `MenuItem`. В классе `MenuItemCollection` реализованы методы, позволяющие добавлять дочерние пункты меню к главному или контекстному меню либо другому пункту.

Некоторые методы:

```
MenuItem Add(string caption)
MenuItem Add(string caption, EventHandler onClick)
MenuItem Add(string caption, MenuItem [] items)
int Add(MenuItem item)
int Add(int index, MenuItem item)
void AddRange(MenuItem[] items)
```

Для определения количества пунктов меню в коллекции можно использовать свойство *Count* класса *MenuItemCollection*.

Рассмотрим использование метода *Add* для создания контекстного меню.

В приведенном примере цвет фона формы меняется в соответствии с выбранным пунктом контекстного меню.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace CntMn
{
    public partial class Form1 : Form
    {
        MenuItem myClr; // пункт меню
        ContextMenu ctmn; // Контекстное меню
        public Form1()
```

```

{
    InitializeComponent();
    ctmn = new ContextMenu();// создание контекстного меню
    //Передаем конструктору делегата функцию обработчик пункта меню
    EventHandler ev = new EventHandler(MnClick);
    //добавляем в меню пункты
    ctmn.MenuItems.Add("Red", ev);
    ctmn.MenuItems.Add("Blue", ev);
    ctmn.MenuItems.Add("Green", ev);

    foreach (MenuItem itm in ctmn.MenuItems)
        itm.RadioCheck = true;
    // Выбираем второй пункт меню
    myClr = ctmn.MenuItems[2];
    myClr.Checked = true;
    //Устанавливаем цвет фона
    this.BackColor = Color.FromName(myClr.Text);
    //связываем с формой контекстное меню
    this.ContextMenu = ctmn;
}
// Это обработчик пункта меню
void MnClick(object sr, EventArgs e)
{
    myClr.Checked = false;//снимаем флажок со старого пункта меню
    myClr = (MenuItem)sr; //получаем ссылку на текущий пункт меню
    myClr.Checked = true;//устанавливаем на нем флажок
    this.BackColor = Color.FromName(myClr.Text);
}
}
}

```


Рассмотрим создание меню с помощью мастеров мастерской разработчика Visual Studio.Net.

Создадим приложение, в котором в меню можно выбрать размер прямоугольника (прямоугольник выводится на экран с запуском приложения) большой или маленький. С помощью контекстного меню можно изменить цвет прямоугольника на красный или синий.

1. Для создания меню и контекстного меню необходимо перенести из ToolBox на поверхность формы MainMenu и ContextMenu.
2. Необходимо добавить в пункты «большой» и «маленький». Для этого выделяем mainMenu1 и добавляем данные пункты.
3. В контекстное меню contextMenu1 добавляем пункты «красный» и «синий» аналогичным способом.
4. Теперь создадим переменные, с которыми связаны все изменения прямоугольника:

```
private int width;  
private int height;  
private Color myColor;
```

5. Инициализируем их в конструкторе Form1():

```
public Form1()  
{  
    InitializeComponent();  
    myColor = Color.Red;  
    width = 10;  
    height = 10;  
}
```

6. В свойствах формы создадим, метод-обработчик события Paint и

отредактируем его, чтобы он имел вид:

```
private void Form1_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
    Graphics dc = e.Graphics;
    Pen myPen = new Pen(myColor);
    dc.DrawRectangle(myPen,50,50,width,height);
}
```

7. Выделим mainMenu1 и двойным щелчком мыши на пункте “Большой” создадим метод-обработчик выбора данного пункта.отредактируем тело метода, после чего оно должно иметь вид:

```
private void menuItem2_Click(object sender, System.EventArgs e)
{
    width = 100;
    height = 100;
    Invalidate();
}
```

8. Аналогично для пункта “Маленький”:

```
private void menuItem3_Click(object sender, System.EventArgs e)
{
    width = 10;
    height = 10;
    Invalidate();
}
```

9. Для пункта contextMenu1 “Красный”:

```
private void menuItem4_Click(object sender, System.EventArgs e)
{
    myColor = Color.Red;
    Invalidate();
}
```

10. Для contextMenu1 “Синий”:

```
private void menuItem5_Click(object sender, System.EventArgs e)
{
    myColor = Color.Blue;
    Invalidate();
}
```

11. Для вывода контекстного меню на экран в свойствах формы создаем метод-обработчик события нажатия клавиши мыши, после редактирования метод должен иметь вид:

```
private void Form1_MouseDown(object sender,
System.Windows.Forms.MouseEventArgs e)
{
    Point myPoint = new Point(e.X,e.Y);
    if(e.Button.ToString() == "Right")
        contextMenu1.Show(this,myPoint);
}
```

12. Компилируем приложение и анализируем полученные результаты.

Вопросы:

1. Какое меню называется главным меню?
2. Что такое контекстное меню?
3. Какие пункты меню называются пунктами верхнего уровня?
4. Какой класс используется для создания главного меню приложения?
5. Можно ли менять главное меню в процессе работы приложения?
6. Как связать контекстное меню с элементом управления?
7. Как определить в меню клавишу быстрого доступа?
8. Как можно сделать пункт меню недоступным?
9. Как можно пункт меню сделать невидимым?
10. Как задать функцию обработчик пункта меню?
11. Как вывести флажок рядом с названием пункта меню?

Задание:

1. Скомпилируйте и выполните приложение по примеру 1. Создайте приложение по примеру 1 с помощью мастеров. Измените, приложение так, чтобы выбранный пункт подменю после срабатывания становился недоступным и вновь становился доступным после срабатывания второго пункта подменю.
2. Скомпилируйте и выполните приложение по примеру 2. Проанализируйте работу приложения. Создайте приложение с помощью мастеров. Добавьте еще несколько пунктов меню для других цветов.
3. Скомпилируйте и выполните приложение по примеру 3. Проанализируйте работу приложения. Определите для пунктов меню клавиши быстрого доступа. Комбинации клавиш быстрого доступа должна отображаться рядом с названием пункта меню.
4. Скомпилируйте и выполните приложение по примеру 3. Проанализируйте работу приложения. Измените, приложение так, чтобы при выборе пункта контекстного меню название цвета (название пункта меню) менялось на некоторый альтернативный. При повторном нажатии название и действие

пункта восстанавливалось.

5. Создайте приложение, в котором при нажатии на один пункт меню, слева направо начинал двигаться шар, а при нажатии на другой пункт меню останавливался.
6. Скомпилируйте и выполните приложение по примеру 3. Проанализируйте работу приложения. Изменить приложение так, чтобы в контекстном меню можно было изменить цвет прямоугольника на зеленый, а в главном меню можно было выбрать средний размер.
7. Скомпилируйте и выполните приложение по примеру 3. Изменить приложение так, чтобы контекстное меню появлялось лишь тогда, когда произведен щелчок правой клавишей мыши в области прямоугольника.
8. Создайте приложение, в котором с помощью меню можно задать направление движения шара, который начинает двигаться при щелчке левой кнопки мыши, а останавливается при щелчке на правой кнопке мыши.
9. Создайте приложение, в котором с помощью меню осуществляется выбор фигуры (эллипс, квадрат, треугольник), а с помощью второго пункта меню выбирается цвет для рисования.
10. Создать приложение, в котором при нажатии на кнопку главное меню приложения, с помощью которого можно нарисовать квадрат и прямоугольник, заменялось на другое главное меню, рисующее окружность и эллипс соответственно.