Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Институт повышения квалификации и переподготовки

Кафедра «Информатика»

В. С. Мурашко

РАЗРАБОТКА ПРИЛОЖЕНИЙ С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ

ПРАКТИКУМ
по дисциплине «Объектно-ориентированное программирование» для слушателей специальности 1-40 01 73 «Программное обеспечение информационных систем» заочной формы обучения

Гомель 2016

УДК 004.43(075.8)
ББК 32.973.2я73
М91

Рекомендовано кафедрой «Информатика» ГГТУ им. П. О. Сухого
(*протокол № 4 от 27.11.2015 г.*)

Рецензент: доц. каф. «Информационные технологии» ГГТУ им. П. О. Сухого канд. техн. наук, доц. *В. В. Комраков*

Мурашко, В. С.

М91 Разработка приложений с графическим интерфейсом : практикум по дисциплине «Объектно-ориентированное программирование» для слушателей специальности 1-40 01 73 «Программное обеспечение информационных систем» заоч. формы обучения / В. С. Мурашко. – Гомель : ГГТУ им. П. О. Сухого, 2016. – 99 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

В практикуме представлено пять работ по дисциплине «Объектно-ориентированное программирование». Даны варианты заданий и порядок выполнения работ с необходимыми методическими указаниями.

Для слушателей по курсу «Объектно-ориентированное программирование» ИПКиП.

УДК 004.43(075.8)
ББК 32.973.2я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Любая методология программирования имеет свою концептуальную основу. Эволюция развития языков программирования направлена на ускорение процесса создания надежных программных средств. Все программы в ЭВМ на самом низком аппаратном уровне приводятся в действие микрокомандами, вырабатываемыми блоком микропрограммного управления (входящим в состав всех современных микропроцессоров). Этот блок для каждой машинной команды вырабатывает набор сигналов-действий, необходимых для ее физической реализации. Символическим аналогом машинного языка является язык ассемблера, с появлением которого и началась история развития языков программирования.

Следующий этап развития языков программирования был связан с созданием технологии структурного программирования. В основе структурного программирования лежит декомпозиция (разбиение на части) программы с целью выделения подзадач и реализация их в виде подпрограмм. Структурное программирование зародилось в начале 70-х годов и до сих пор является основой в большом количестве проектов. С помощью структурного программирования можно разработать очень большую программу, однако это потребует усилий группы профессиональных программистов.

В 80-х годах стали появляться первые коммерческие системы разработки, в которых была реализована новая парадигма программирования, так называемый объектный подход, что позволило резко повысить производительность труда программистов. Подход был основан на понятии объекта, типа данных, в котором сочетаются как свойства, сгруппированные данные (пример – поля в записи), так и методы их обработки (подпрограммы). Фактически объект стал отражать реальные и даже абстрактные понятия окружающего мира. Благодаря этому теперь удается выполнить проектирование программ, основываясь на понятии объекта, что значительно проще и быстрее, чем раньше. Работать с привычными понятиями человеку легче, нежели с абстрактными числами. При этом специалистам удалось выделить большой набор объектов, которые нужны при создании самых разных программ. Эти объекты используются повторно без расходования времени на их программирование.

Лабораторные работы, описанные в данном практикуме, выполняются при помощи объектно-ориентированного языка C#, входящего в состав пакета Microsoft Visual Studio 2008 и выше. Язык C# как средство обучения программированию обладает рядом достоинств. Он хорошо организован, строг, большинство его конструкций логичны и понятны, а развитые средства диагностики и редактирования кода делают процесс программирования эффективным.

1 РАЗРАБОТКА ПРИЛОЖЕНИЯ С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ

Цель: *получить навыки разработки программ, использующих формы, элементы управления и принципы событийно-управляемого программирования.*

1.1 Краткие теоретические сведения

Создание приложения состоит из двух этапов.

1. Визуальное проектирование – создание интерфейса приложения (конструирование форм).
2. Разработка и реализация алгоритма решения задачи путем написания процедур обработки событий.

При создании приложений наиболее часто используются следующие события:

- *Activated* **–** получение формой фокуса ввода.
- *Click, DoubleClick* – одинарный и двойной щелчки мышью
- *Closed* – закрытие формы.
- *KeyPress* – нажатие клавиши, имеющей ASCII-код.
- *MouseMove* **–** перемещение мыши и др.

Методы класса *Form:*

- *Activate( )* – активизирует форму и передает ей фокус ввода. Например, если *f* – объект класса *Form* (или производного от него), то активизировать форму *f* можно с помощью оператора *f.Activate()*. При этом, если форма не была отображена на экране, то она остается невидимой.
- *Close( )* – закрывает форму. Например, *f.Close( )*. Для текущей формы *this.Close()* или просто *Close()*.
- *Show()* – отображает форму в немодальном режиме (это не дополнительный метод, а унаследованный от класса Control). Например, *f.Show()*.
- *ShowDialog( )* – отображает форму как диалоговое окно. Окно отображается в модальном режиме. При закрытии окна этим методом формируется результат выполнения из значений перечисления *DialogResult (OK, Cancel)*. Предполагается, что в окне есть две кнопки, одна из которых зафиксирована в свойстве *AcceptButton***,** а другая – в свойстве *CancelButton*.

Чтобы создать новую форму, нужно выполнить команду меню *Проект – Добавить форму Windows*

Чтобы иметь возможность доступа к компонентам созданной формы, эти компоненты должны иметь спецификатор доступа *public*. Поэтому в окне свойств для этих компонентов необходимо установить значение «свойства» *Modifiers - public*.

Прежде чем использовать элементы управления созданной формы, нужно создать объект созданного класса, а затем по имени этого объекта обращаться к элементам управления, а также ко всем элементам класса.

Например,

```
Form2   f2 = new Form2( );
f2.textBox1.Text = "Привет!";   f2.ShowDialog( );
```

Ввод одномерного массива (элементы управления *RichTextBox* и *DataGridView*)**.**

Для ввода массива можно использовать компонент *RichTextBox* (многострочный текстовый редактор).

Основные свойства:

- *Text* – содержит редактируемый текст (тип *string*);
- *Lines* – содержит массив строк типа *string*, элементами которого являются строки редактируемого текста;
- *ReadOnly* – устанавливает доступ только для чтения.

На рисунке 1.1 приведен пример ввода одномерного массива в виде столбца, а на рисунке 1.2 – вид компонента с введенными данными. В массив будут записаны только числа.

```
string[ ]   s = richTextBox1.Lines;
int  n = s.Length;
int[ ]  x =  new  int[n];
int j = 0;
for (int i = 0; i < n; i++)
{ try
{x[j] = Convert.ToInt32(s[i]);
j=j+1;
}
catch  {}
}
int[]   y = new int[j];
for (int i = 0; i < j; i++)    y[i] = x[i];
```

Рисунок 1.1 – Ввод одномерного массива

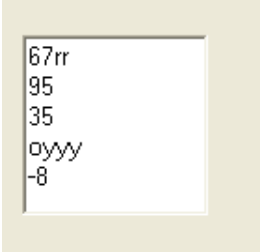


Рисунок 1.2– Вид RichTextBox

Для ввода массива можно использовать также компонент класса *DataGridView***,** который представляет собой таблицу для редактирования данных.

Класс *DataGridView* имеет два индексатора. Оба они возвращают значение типа *DataGridViewCell* – объект, содержащий информацию об ячейке таблицы**,** и имеют два индекса.

В первом из них оба индекса *целые* и обозначают номер столбца и номер строки. Пример обращения:

```
dataGridView1[0, 1]
```

Во втором индексаторе первый индекс имеет тип *string* и обозначает имя столбца, а второй индекс имеет тип *int* и обозначает номер строки. Пример обращения:

```
dataGridView1["Column1", 1]
```

Чтобы получить доступ к значению в ячейке таблицы, можно использовать свойство *Value* класса *DataGridViewCell* **(**тип *string*):

```
dataGridView1[0, 1].Value
```

Основные свойства *DataGridView:*

- *RowCount* – число строк в таблице. (Не отражается в окне свойств).
- *ColumnCount* – число столбцов в таблице.
- *RowHeadersVisible* – определяет, выводится ли столбец с заголовками строк (*true* – выводится, *false* – нет).
- *ColumnHeadersVisible* – определяет, выводится ли строка с заголовками столбцов (*true* – выводится, *false* – нет).
- *CurrentCell* – делает некоторую ячейку активной или обращается к активной ячейке. Тип этого свойства *DataGridViewCe**ll***.

Пример, иллюстрирующий, как сделать какую- то ячейку текущей:

```
dataGridView1.CurrentCell = dataGridView1[0, 1];
```

Чтобы получить доступ к значению, размещенному в текущей ячейке, можно использовать свойство *Value* класса *DataGridViewCell*. Например: `dataGridView1.CurrentCell.Value`

Рассмотрим пример, демонстрирующий возможность ввода одномерного массива с помощью компонента *DataGridView*.

Пусть компонент при выполнении программы выглядит на рисунке 1.3 следующим образом (ввод массива целых чисел в столбец):

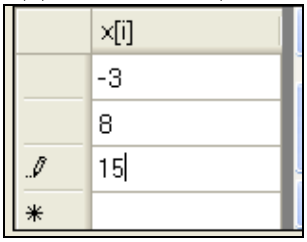


Рисунок 1.3 – Ввод массива целых чисел в столбец.

Тогда фрагмент программы для считывания данных из таблицы в массив может выглядеть как рисунке 1.4.

```
int  n = dataGridView1.RowCount-1;
int[ ]  x = new   int[n];
for (int i = 0; i < x.Length; i++)
   x[i] = Convert.ToInt32(dataGridView1[0, i].Value);
```

Рисунок 1.4 – Считывания данных из таблицы в массив

Можно организовать ввод массива в строку, при этом размерность массива можно вводить с клавиатуры (рисунок 1.5)

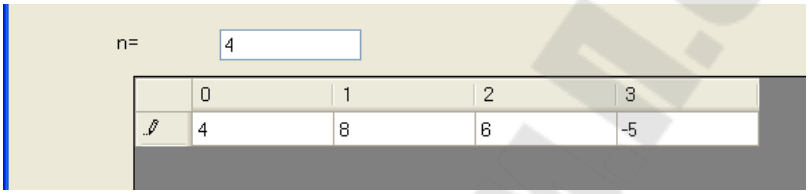


Рисунок 1.5 – Ввод массива в строку

Чтобы при вводе элемента массива не создавалась новая строка нужно для свойства *AllowUserToAddRows* таблицы установить значение *false*.

Чтобы при вводе размерности массива, автоматически изменялось количество столбцов в таблице, и выводились номера элементов в качестве заголовков столбцов, нужно создать обработчик события *TextChanged* компонента *TextBox* (рис.1.6).

```
private void textBox1_ModifiedChanged(object sender, EventArgs
e)
{
            if (textBox1.Text != "")
dataGridView1.ColumnCount = Convert.ToInt32(textBox1.Text);
 dataGridView1.RowCount = 1;
for (int i = 0; i < dataGridView1.ColumnCount; i++)
dataGridView1.Columns[i].HeaderText = i.ToString();}
```

Рисунок 1.6 – Обработчик события *TextChanged* компонента *TextBox*

Тогда фрагмент программы для ввода одномерного массива в строку может выглядеть на рисунке 1.7.

```
private void button1_Click(object sender, EventArgs e)
{int n = Convert.ToInt32(textBox1.Text);
int[] x = new int[n];
for (int i = 0; i < x.Length; i++)
x[i] = Convert.ToInt32(dataGridView1[i, 0].Value);}
```

Рисунок 1.7 – Ввод одномерного массива в строку

Для ввода матрицы с помощью компонента *DataGridView* (рис. 1.8) понадобятся два компонента для ввода размерности. Для каждого из них создать обработчик события *TextChanged* (рис. 1.9).

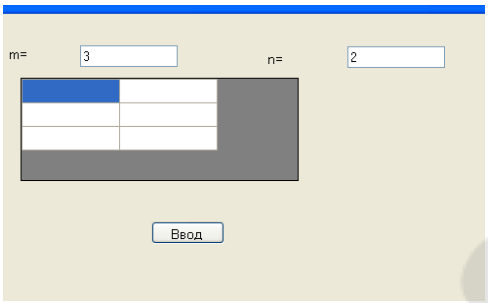


Рисунок 1.8 – Ввод матрицы

```
private void textBox1_ModifiedChanged(object sender,EventArgs e)
        {
if (textBox1.Text != "")
dataGridView1.RowCount = Convert.ToInt32(textBox1.Text);
        }
private void textBox2_TextChanged(object sender, EventArgs e)
        {
if (textBox2.Text != "")
dataGridView1.ColumnCount = Convert.ToInt32(textBox2.Text);
        }
```

Рисунок 1.9 – Обработчики события *TextChanged* для ввода размерности матрицы

При этом нужно установить свойства компонента *dataGridView1*:

- *AllowUserToAddRows – false;*
- *RowHeadersVisible – false;*
- *ColumnHeadersVisible – false.*

Тогда фрагмент программы для ввода матрицы целых чисел может выглядеть как на рисунке 1.10.

```
private void button1_Click(object sender, EventArgs e)
{
int m = Convert.ToInt32(textBox1.Text);
int n = Convert.ToInt32(textBox2.Text);
int[,] x = new int[m,n];
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
x[i,j] = Convert.ToInt32(dataGridView1[j, i].Value);
        }
```

Рисунок 1.10 – Фрагмент программы для ввода матрицы целых чисел

Вывод массивов с помощью компонента *DataGridView* представлен на рисунке 1.11.

```
dataGridView2.RowCount = x.GetLength(0);
dataGridView2.ColumnCount = x.GetLength(1);
for (int  i = 0; i < x.GetLength(0);  i++)
                for (int  j = 0; j < x.GetLength(1);  j++)
dataGridView2[j, i].Value = x[i, j];
```

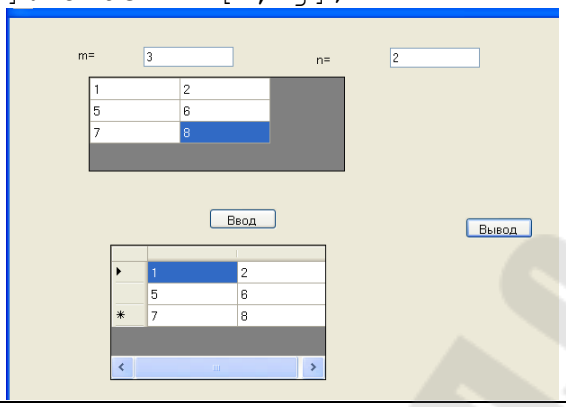


Рисунок 1.11 – Вывод матрицы целых чисел

Компонент *RadioButton* представляет собой переключатель, предназначенный для выбора одного из нескольких взаимоисключающих вариантов.

На форму нужно поместить как минимум два таких компонента.

Свойство *Checked* – *true*, если компонент выбран (переключатель включен), в противном случае *false*.

Среди переключателей только у одного это свойство может быть *true*.

В программе для выбора действия с помощью переключателя *RadioButton* нужно использовать обращение к свойству *Checked*.

Например:

```
if (radioButton1.Checked)
        {textBox1.Text = "Все на экзамен!"; }
```

Компонент *ListBox* представляет список, с помощью которого можно выбрать один или несколько пунктов.

Свойства:

- *Items* – содержит набор строк – названий пунктов в списке;
- *SelectedIndex* – содержит индекс элемента списка, имеющего фокус ввода.

Свойства раскрывающийся списка *ComboBox* (рис. 1.12):

- *Items* – содержит набор строк – названий пунктов в списке;
- *SelectedIndex* – содержит индекс элемента списка, имеющего фокус ввода.

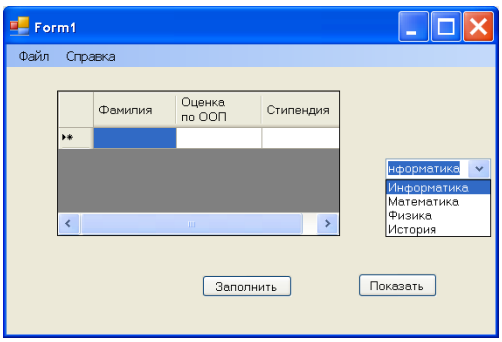


Рисунок 1.12 – Вид формы с *ComboBox*

Создание и использование диалоговых окон.

Примеры диалоговых окон:

- окно сохранения файла;
- окно открытия файла;
- окно установки параметров шрифта и т.д.

Работа с диалоговыми окнами осуществляется в три этапа:

1. Размещение соответствующего компонента на форме (в момент работы программы этот компонент не виден, видно только создаваемое диалоговое окно).
2. Вызов в программе метода *ShowDialog()*. Именно после обращения к этому методу на экране появляется соответствующее диалоговое окно, которое является модальным.
3. Анализ результата вызова метода *ShowDialog()* и использование в программе результатов диалога. Метод возвращает результат типа *DialogResult* (перечисление): *Abort*, *OK*, *No*, *Cancel* и др.

Диалоги:

- *SaveFileDialog* – диалог сохранения файла;
- *OpenFileDialog* - диалог открытия файла.

Фрагмент программы для открытия файла представлен на рисунке 1.13

```
if (openFileDialog1.ShowDialog( ) = = DialogResult.OK)
  {StreamReader f = new StreamReader(openFileDialog1.FileName);
     string s = f.ReadToEnd();
      f.Close();}
```

Рисунок 1.13 – Открытие файла

Пример 1.1 Написать Windows-приложение, работающее под управлением меню:

Файл

Новый

Открыть

Сохранить
Выход
Обработка матрицы
Сумма…
Произведение
Об авторе

Для решения задачи создать класс «Матрица», в котором описать следующие элементы: закрытое поле – матрица целых чисел, индексатор для доступа к элементам поля-массива, конструктор с параметрами, перегруженные методы для поиска суммы всех элементов матрицы и суммы элементов заданной строки, метод для вычисления произведения элементов.

При выполнении команды *Файл* - *Новый* открывается таблица *DataGridView* для ввода матрицы, появляется кнопка *Сохранить*, кнопка *Обработка* и панель с переключателями для выбора вида вычислений в матрице. Пользователь получает возможность ввода матрицы.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла открывается таблица *DataGridView* с загруженной матрицей, появляется кнопка *Обработка* и панель с переключателями для выбора вида вычислений в матрице.

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. После ввода имени матрица сохраняется в соответствующем файле. То же самое происходит при нажатии кнопки *Сохранить*.

При выполнении команды *Файл – Сумма…* появляется окно с переключателями, определяющими, какой вид суммы нужно найти.

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

Вид главного окна при запуске (рис. 1.14):

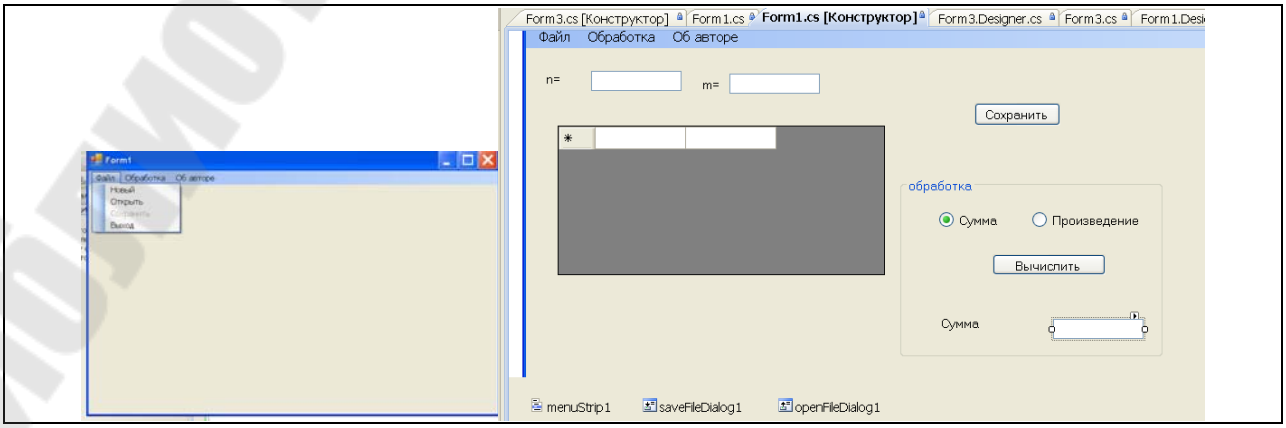


Рисунок 1.14 – Вид главного окна при запуске

Вид окна после выполнения команды *Файл-Новый* (рис.1.15):

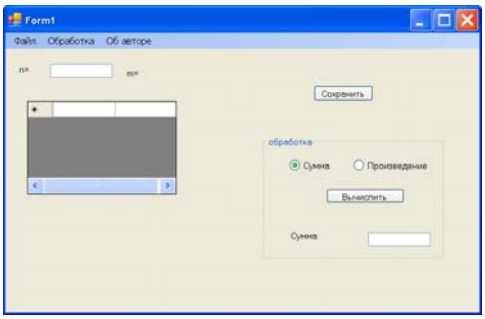


Рисунок 1.15 – Вид окна после выполнения команды *Файл-Новый*

Окно, открываемое после выполнения команды меню об авторе (рис.1.16):

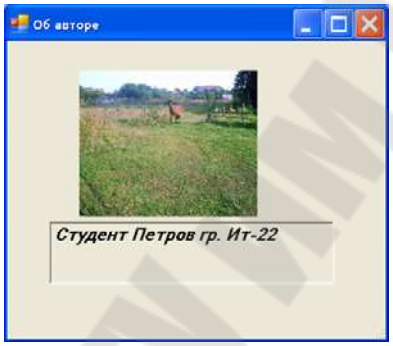


Рисунок 1.16 – Вид окна об авторе

Окно после выполнения команды *Обработка-Сумма…* или нажатия кнопки *Вычислить* при установленном переключателе *Сумма* (рис. 1.17).

Свойства элементов управления:

- *button1 – Text* – Выполнить*; DialogResult – OK;*
- *button2 – Text* – *Отмена*; *DialogResult – Cancel.*

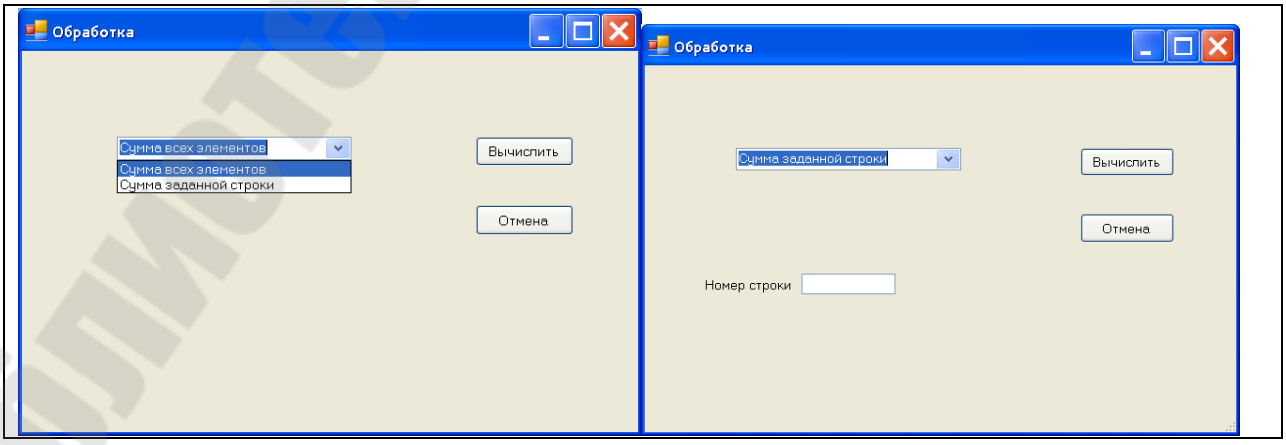


Рисунок 1.17 – Вид окна «Обработка»

Создадим класс *Матрица* в файле form1.cs (рис. 1.18). Этот класс должен располагаться **после класса Form1**. А также все необходимые обработчики событий (рис. 1.19).

```
class Matrica
    { int[,] a;
public Matrica(int n,int m)
        {
            a = new int[n, m];
        }
public Matrica(DataGridView DgV)
{ int   n = DgV.RowCount;
  int   m = DgV.ColumnCount;
a = new int[n,m]; }
public   int   this[int i, int j]
        {
            get { return a[i, j]; }
            set { a[i, j] = value; }
        }
public   int   GetLength(int p)
        { return a.GetLength(p); }
public void read( DataGridView DgV)
{
        for (int i = 0; i < a.GetLength(0); i++)
   for (int j = 0; j < a.GetLength(1); j++)
a[i, j] = Convert.ToInt32(DgV[j, i].Value);
    }
public void write(DataGridView DgV)
{ DgV.RowCount = a.GetLength(0);
    DgV.ColumnCount = a.GetLength(1);
for (int i = 0; i < a.GetLength(0); i++)
    for (int j = 0; j < a.GetLength(1); j++)
                DgV[j, i].Value = a[i, j];}
public int proizv()
{
    int p = 1;
    for (int i = 0; i < a.GetLength(0); i++)
        for (int j = 0; j < a.GetLength(1); j++)
            p = p * a[i, j];
    return p;}
public int summa( )
        { int S = 0;
        foreach (int x in a) S = S + x;
        return S;
        }
public int summa(int   ns)
        {   int S=0;
        for (int j = 0; j < a.GetLength(1);j++ ) S = S + a[ns,
j];
            return S;}}
```

Рисунок 1.18 – Класс *Матрица*

```
private void Save(Matrica A)
        {
if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
StreamWriter f = new StreamWriter(saveFileDialog1.FileName);
for (int i = 0; i < A.GetLength(0); i++)
{
for (int j = 0; j < A.GetLength(1); j++)
f.Write(A[i, j] + " ");
f.WriteLine();
                }
f.Close();
            }
        }
private void новыйToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            label1.Visible = true;
            label2.Visible = true;
            textBox1.Visible = true;
            textBox2.Visible = true;
            groupBox1.Visible = true;
            button1.Visible = true;
            dataGridView1.Visible = true;
открытьToolStripMenuItem.Enabled = true;
        }
private void обАвтореToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            (new Form2()).Show();
        }

   private void button1_Click(object sender, EventArgs e)
        {
Matrica A = new Matrica(dataGridView1);
            A.read(dataGridView1);
            Save(A);

        }
//Не предусмотрено исключение -если не цифра!!!
private void textBox1_TextChanged(object sender, EventArgs e)
    {if (textBox1.Text != "")
dataGridView1.RowCount = Convert.ToInt32(textBox1.Text) + 1;
dataGridView1.AllowUserToAddRows = false;
        }

        private void textBox2_TextChanged(object sender,
EventArgs e)
            {if (textBox2.Text != "")
dataGridView1.ColumnCount = Convert.ToInt32(textBox2.Text);
        }
```

Рисунок 1.19 – Методы и обработчики событий класса *Form1*

```
private void выходToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Close();
        }
private void открытьToolStripMenuItem_Click(object sender,
EventArgs e)
        {
if (openFileDialog1.ShowDialog() == DialogResult.OK)
                {label1.Visible = true;
                label2.Visible = true;
                textBox1.Visible = true;
                textBox2.Visible = true;
                groupBox1.Visible = true;
                button1.Visible = true;
                dataGridView1.Visible = true;
сохранитьToolStripMenuItem.Enabled = true;
StreamReader f1 = new StreamReader(openFileDialog1.FileName);
                string s = f1.ReadLine();
                int j = 0;
                string[] ss = s.Split(' ');
  dataGridView1.ColumnCount = ss.Length - 1;
                while (s != null)
                {
                    s = f1.ReadLine();
                    j++;
                }
dataGridView1.RowCount = j;
                f1.Close();
f1 = new StreamReader(openFileDialog1.FileName);
                s = f1.ReadLine(); j = 0;
                while (s != null)
                {
                    ss = s.Split(' ');
for (int i = 0; i < dataGridView1.ColumnCount; i++)
                    {
dataGridView1[i, j].Value = ss[i];
                    }
                    s = f1.ReadLine();
                    j++;
                }
                f1.Close();
            }
        }
private void сохранитьToolStripMenuItem_Click(object sender,
EventArgs e)
        {
 Matrica A = new Matrica(dataGridView1);
            A.read(dataGridView1);
            Save(A);}
```

Продолжение рисунка 1.19

```
private void суммаToolStripMenuItem_Click(object sender,
EventArgs e)
        {
Matrica A = new Matrica(dataGridView1);
            A.read(dataGridView1);
            textBox3.Text = "";
            int S = 0;
                Form3 f3 = new Form3();
if (f3.ShowDialog() == DialogResult.OK)
                {
if (f3.comboBox1.SelectedIndex == 0)
                    {
                        S = A.summa();
                        label3.Text = "Сумма";
textBox3.Text = S.ToString();
                    }
else
if (f3.textBox1.Text != "")
  {int k=0;
 k=Convert.ToInt32(f3.textBox1.Text);
if (k <= A.GetLength(0))
{
S = A.summa(k - 1);
label3.Text = "Сумма";
textBox3.Text = S.ToString();
            }
else MessageBox.Show("Строка "+k.ToString()+"  не существует");
}
 else
  MessageBox.Show("Не указан номер строки " , "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                }
            }

private void произведениеToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Matrica A = new Matrica(dataGridView1);
            A.read(dataGridView1);
            int p = A.proizv();
            label3.Text = "Произведение";
            textBox3.Text = p.ToString();
        }
```

Продолжение рисунка 1.19

```
private void button2_Click(object sender, EventArgs e)
        {
 Matrica A = new Matrica(dataGridView1);
            A.read(dataGridView1);
            textBox3.Text = "";
            int S = 0;
            if (radioButton1.Checked)
            {
                Form3 f3 = new Form3();
 if (f3.ShowDialog() == DialogResult.OK)
                {
 if (f3.comboBox1.SelectedIndex == 0)
                    {
                        S = A.summa();
                        label3.Text = "Сумма";
 textBox3.Text = S.ToString();
                    }
else
  if (f3.textBox1.Text != "")
                        {int k = 0;
k = Convert.ToInt32(f3.textBox1.Text);
     if (k <= A.GetLength(0))
  {
S = A.summa(k - 1);
label3.Text = "Сумма";
textBox3.Text = S.ToString();
                            }
       else
MessageBox.Show("Строка " + k.ToString() + "  не существует");}

 MessageBox.Show("Не указан номер строки ", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                }
            }
else
            {
                int p = A.proizv();
                label3.Text = "Произведение";
                textBox3.Text = p.ToString();
            }
        }
```

Продолжение рисунка 1.19

1.2 Варианты заданий

Вариант 1

Написать Windows-приложение, работающее под управлением меню:

Файл
  Новый
  Открыть
  Сохранить
  Выход
Анализ текста
  Количество слов
  Количество предложений
  Выбор телефонов
Об авторе

При выполнении команды *Файл - Новый* открывается окно *RichTextBox*, появляется кнопка *Сохранить* и панель с кнопкой *Анализ* и переключателями для выбора вида анализа текста:

Количество слов
Количество предложений
Выбор телефонов

Пользователь получает возможность ввода текста.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла открывается окно *RichTextBox* с загруженным текстом, появляется кнопка *Сохранить* и панель с кнопкой *Анализ* и переключателями для выбора вида анализа текста**.**

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. То же самое происходит при нажатии кнопки Сохранить**.**

При выполнении команды *Анализ текста - Количество слов* определяется количество слов в тексте, начинающихся с заданной буквы (указать в текстовом окне). Команда дублируется кнопкой *Анализ.*

При выполнении команды *Анализ текста - Количество предложений о*пределяется количество цитат в тексте, т.е. предложений, заключенных в кавычки. Команда дублируется кнопкой **Анализ.**

При выполнении команды *Анализ текста - Выбор телефонов* формируется массив из номеров телефонов, присутствующих в тексте и выводится в таблицу *DataGridView*. Команда дублируется кнопкой *Анализ.*

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

Вариант 2

Написать Windows-приложение, работающее под управлением меню:

Файл
  Новый
  Открыть
  Сохранить
  Выход
Обработка массива
  Минимум
  Максимум…
  Сортировка
Об авторе

Для решения задачи создать класс «Одномерный массив», в котором описать следующие элементы: закрытое поле – массив целых чисел, свойство для определения длины массива, индексатор для доступа к элементам поля-массива, конструктор с параметрами, перегруженные методы для поиска максимального элемента во всем массиве и для поиска максимального элемента в части массива, ограниченной начальным и конечным значениями индекса, передаваемых в метод в качестве параметров, методы для поиска минимального элемента и сортировки массива.

При выполнении команды *Файл - Новый* открывается окно *RichTextBox*, появляется кнопка *Сохранить***,** кнопка *Обработка* и раскрывающийся список для выбора вида обработки массива. Пользователь получает возможность ввода одномерного массива.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла открывается окно редактора *RichTextBox* с загруженным массивом, появляется кнопка *Обработка* и раскрывающийся список для выбора вида обработки массива**.**

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. После ввода имени массив сохраняется в соответствующем файле. То же самое происходит при нажатии кнопки *Сохранить*.

При выполнении команды *Обработка массива – Максимум…* появляется окно с переключателями, определяющими, в какой части массива осуществляется поиск максимума.

При выполнении команды *Обработка массива –Сортировка* исходный массив и отсортированный выводятся в новом окне в таблицах *DataGridView*.

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

Вариант 3

Написать Windows-приложение, работающее под управлением меню:

Файл
  Новый
  Открыть
  Сохранить
  Выход
Обработка матрицы
  Сумма
  Произведение…
Уплотнить
Об авторе

Для решения задачи создать класс «Матрица», в котором описать следующие элементы: закрытое поле – матрица целых чисел, индексатор для доступа к элементам поля-массива, конструктор с параметрами, методы для поиска суммы элементов матрицы, перегруженные методы для вычисления произведения элементов, принадлежащих заданному интервалу (границы интервала передаются в метод в качестве параметров) и для вычисления произведения элементов, расположенных выше побочной диагонали (с проверкой, является ли матрица квадратной); уплотнить заданную матрицу, удаляя из нее строки и столбцы заполненные нулями.

При выполнении команды *Файл - Новый* открывается таблица *DataGridView* для ввода матрицы, появляется кнопка *Сохранить***,** кнопка *Обработка* и панель с переключателями для выбора вида вычислений в матрице. Пользователь получает возможность ввода матрицы.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла открывается таблица *DataGridView* с загруженной матрицей, появляется кнопка *Обработка* и панель с переключателями для выбора вида вычислений в матрице.

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. После ввода имени матрица сохраняется в соответствующем файле. То же самое происходит при нажатии кнопки *Сохранить*.

При выполнении команды *Обработка матрицы – Произведение…* появляется окно с переключателями, определяющими, какой вид произведения нужно найти.

При выполнении команды *Обработка матрицы – Уплотнить* – исходная матрица и уплотненная выводятся в новом окне в таблицах *DataGridView.*

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

Вариант 4

Написать Windows-приложение, работающее под управлением меню:

Файл
- Новый
- Открыть
- Сохранить
- Выход

Обработка массива
- Сумма отрицательных элементов
- Произведение…
- Сортировка

Об авторе

Для решения задачи создать класс «Одномерный массив», в котором описать следующие элементы: закрытое поле – массив целых чисел, свойство для определения длины массива, индексатор для доступа к элементам поля-массива, конструктор с параметрами, перегруженные методы вычисления произведения элементов с четными номерами, для вычисления произведения элементов, принадлежащих заданному интервалу (границы интервала передаются в метод в качестве параметров), сортировки массива по следующему правилу: сначала располагаются все положительные элементы в порядке возрастания, а затем отрицательные – в порядке убывания (элементы равные нулю считаются положительными) .

При выполнении команды *Файл - Новый* открывается таблица *DataGridView*, появляется кнопка *Сохранить***,** кнопка *Обработка* и список для выбора вида обработки массива. Пользователь получает возможность ввода одномерного массива.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла открывается окно редактора *RichTextBox* с загруженным массивом, появляется кнопка *Обработка* и список для выбора вида обработки массива**.**

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. После ввода имени массив сохраняется в соответствующем файле. То же самое происходит при нажатии кнопки *Сохранить*.

При выполнении команды *Обработка массива – Произведение…* появляется окно с переключателями, определяющими вид произведения, и полями для ввода границ интервала.

Вывод результатов и исходного массива осуществляется в новом окне.

При выполнении команды *Обработка массива –Сортировка* исходный массив и отсортированный выводятся в новом окне в таблицах *DataGridView***.**

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

Вариант 5

Написать Windows-приложение, работающее под управлением меню:

Файл
  Новый
  Открыть
  Сохранить
  Выход
Обработка матрицы
   Максимум
   Произведение…
  Упорядочить
 Об авторе

Для решения задачи создать класс «Матрица», в котором описать следующие элементы: закрытое поле – матрица целых чисел, индексатор для доступа к элементам поля-массива, конструктор с параметрами, методы для поиска местоположения максимального элемента матрицы, перегруженные методы для вычисления произведения элементов, принадлежащих заданному интервалу (границы интервала передаются в метод в качестве параметров) и для вычисления произведения элементов, расположенных ниже главной диагонали (с проверкой, является ли матрица квадратной); упорядочить строки матрицы по возрастанию количества одинаковых элементов в каждой строке.

При выполнении команды *Файл - Новый* открывается окно редактора *RichTextBox* для ввода матрицы, появляется кнопка

Сохранить, кнопка *Обработка* и список для выбора вида вычислений в матрице. Пользователь получает возможность ввода и обработки матрицы.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла открывается таблица *DataGridView* с загруженной матрицей, появляется кнопка *Обработка* и список для выбора вида вычислений в матрице.

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. После ввода имени матрица сохраняется в соответствующем файле. То же самое происходит при нажатии кнопки *Сохранить*.

При выполнении команды *Обработка матрицы–Произведение…* появляется окно с раскрывающимся списком, определяющим какой вид произведения нужно найти, и полями для ввода границ интервала.

При выполнении команды *Обработка матрицы – Максимум* максимальный элемент матрицы в таблице подсвечивается красным цветом.

При выполнении команды *Обработка матрицы – Упорядочить* – исходная матрица и упорядоченная выводятся в новом окне в таблицах *DataGridView.*

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

Вариант 6

Написать Windows-приложение, работающее под управлением меню:

Файл
  Новый
  Открыть
  Сохранить
  Выход
Обработка
  Средний балл
  Лучший студент…
Сортировка…
Об авторе

Для решения задачи создать класс *Student* содержащий следующие элементы: закрытые поля фамилия, группа, успеваемость (массив), индексатор для доступа к элементам массива оценок, метод для определения среднего балла за сессию.

При выполнении команды *Файл - Новый* появляется таблица *DataGridView* для ввода массива объектов класса *Student*, появляется кнопка *Сохранить***,** кнопка *Средний балл* и кнопка *Лучший студент*. Пользователь получает возможность ввода и обработки информации.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла информация из него считывается в массив объектов и выводится в таблицу *DataGridView,* появляются кнопки *Средний балл, Лучший студент, Сортировка….*

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. После ввода имени информация сохраняется в соответствующем файле. То же самое происходит при нажатии кнопки *Сохранить***.**

При выполнении команды *Обработка – Средний балл* в отдельном окне выводится таблица со списком студентов и средним баллом для каждого студента.

При выполнении команды *Обработка– Лучший студент* в отдельном окне вводится название группы для поиска, после чего в исходной таблице информация о студенте с самым высоким средним баллом по заданной группе подсвечивается красным цветом.

При выполнении команды Обработка– Сортировка… появляются переключатели: сортировать по среднему баллу в каждой группе или во всем группам, после чего в новом окне выводится в таблице *DataGridView* отсортированный массив: ФИО студента, группа, средний балл

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

Вариант 7

Написать Windows-приложение, работающее под управлением меню:

Файл
- Новый
- Открыть
- Сохранить
- Выход

Обработка
- Средняя нагрузка
- Поиск
- Сортировка…

Об авторе

Для решения задачи создать класс *Prepodavatel*, содержащий следующие элементы: закрытые поля фамилия, кафедра, нагрузка в часах за 10 месяцев (массив), индексатор для доступа к элементам массива нагрузки, метод для определения средней нагрузки за учебный год.

При выполнении команды *Файл - Новый* появляется окно редактора *RichTextBox* для ввода массива объектов класса *Prepodavatel,* появляется кнопка *Сохранить***,** кнопка *Обработка* и переключатели *Средняя нагрузка* и *Поиск*. Пользователь получает возможность ввода и обработки информации.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла информация из него считывается в массив объектов и выводится в таблицу *DataGridView*, появляются кнопки *Средняя нагрузка* и *Поиск*.

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. После ввода имени информация сохраняется в соответствующем файле. То же самое происходит при нажатии кнопки *Сохранить*.

При выполнении команды *Обработка – Средняя нагрузка* определяется средняя нагрузка по заданной кафедре.

При выполнении команды *Обработка– Поиск* в отдельном окне вводится название кафедры и выбирается месяц для поиска, после чего выводится информация о нагрузке преподавателей заданной кафедры за указанный месяц.

При выполнении команды *Обработка– Сортировка…* в отдельном окне появляются переключатели: *по кафедрам*, *по вузу* после чего от выбора в таблицу *DataGridView* выводится список преподавателей в порядке возрастания средней нагрузки или на отдельно взятой кафедре или во всему вузу.

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

Вариант 8

Написать Windows-приложение, работающее под управлением меню:

Файл
  Новый
  Открыть
  Сохранить
  Выход
Анализ текста

Количество слов
Количество предложений
Выбор интернет-адресов
Об авторе

При выполнении команды *Файл - Новый* открывается окно редактора текста, появляется кнопка *Сохранить* и панель с кнопкой *Анализ* и раскрывающимся списком для выбора вида анализа текста. Пользователь получает возможность ввода текста.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла открывается окно редактора с загруженным текстом, появляется панель с кнопкой *Анализ* и переключателями для выбора вида анализа текста**.**

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. То же самое происходит при нажатии кнопки *Сохранить***.**

При выполнении команды *Анализ текста - Количество слов* определяется количество слов в тексте, начинающиеся и оканчивающиеся на гласные. Команда дублируется кнопкой *Анализ***.**

При выполнении команды *Анализ текста - Количество предложений* определяется количество предложений в тексте, не содержащих запятых. Команда дублируется кнопкой *Анализ.*

При выполнении команды *Анализ текста - Выбор интернет-адресов* формируется массив из интернет-адресов, присутствующих в тексте, и выводится в таблицу *DataGridView*. Команда дублируется кнопкой *Анализ.*

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

Вариант 9

Написать Windows-приложение, работающее под управлением меню:

Файл
Новый
Открыть
Сохранить
Выход
Обработка массива
Сумма
Количество…
Сортировка…
Об авторе

Для решения задачи создать класс «Одномерный массив», в котором описать следующие элементы: закрытое поле – массив целых чисел, свойство для определения длины массива, индексатор для доступа к элементам поля-массива, конструктор с параметрами, перегруженные методы для поиска количества отрицательных элементов во всем массиве и для поиска количества элементов, лежащих в заданном диапазоне, передаваемых в метод в качестве параметров; метод для вычисления суммы модулей элементов массива, расположенных после первого элемента, равного нулю; методы сортировки массива.

При выполнении команды *Файл - Новый* появляется таблица *DateGridView*, появляется кнопка *Сохранить***,** кнопка *Обработка* и панель с переключателями для выбора вида обработки массива. Пользователь получает возможность ввода и обработки одномерного массива.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла открывается таблица *DateGridView* с загруженным массивом, появляется кнопка *Обработка* и панель с переключателями для выбора вида обработки массива**.**

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. После ввода имени массив сохраняется в соответствующем файле. То же самое происходит при нажатии кнопки *Сохранить.*

При выполнении команды *Обработка массива – Количество…* появляется меню, определяющее, диапазон попадания элементов.

При выполнении команды *Обработка массива – Сортировка* заменить все отрицательные элементы массива их квадратами и упорядочить элементы массива по выбранному с помощью раскрывающегося списка виду сортировки: по убыванию/ по возрастанию. Результаты: исходный и отсортированный массив отобразить в новом окне в окнах редактора *RichTextBox.*

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

Вариант 10

Написать Windows-приложение, работающее под управлением меню:

Файл

Новый

Открыть

Сохранить
Выход
Обработка матрицы
Сумма
Максимум
Сортировка…
Об авторе

Для решения задачи создать класс «Матрица», в котором описать следующие элементы: закрытое поле – матрица целых чисел, индексатор для доступа к элементам поля-массива, конструктор с параметрами, метод для вычисления суммы элементов в тех столбцах, которые не содержат отрицательных элементов; метод для поиска максимума среди сумм элементов диагоналей, параллельных главной диагонали матрицы (проверить квадратная ли матрица); перегруженные методы для выполнения сортировки матрицы: перестановка строк матрицы в порядке убывания элементов заданного столбца (параметр – номер столбца) и перестановка столбцов матрицы в порядке возрастания элементов главной диагонали без параметров (проверять, является ли матрица квадратной).

При выполнении команды *Файл - Новый* открывается таблица *DataGridView* для ввода матрицы, появляется кнопка *Сохранить***,** кнопка *Обработка* и список для выбора вида обработки матрицы. Пользователь получает возможность ввода матрицы.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла открывается окно редактора *RichTextBox* с загруженной матрицей, появляется кнопка *Обработка* и список для выбора вида обработки матрицы.

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. После ввода имени матрица сохраняется в соответствующем файле. То же самое происходит при нажатии кнопки *Сохранить.*

При выполнении команды *Обработка матрицы – Сумма* результат выводится в текстовое окно.

При выполнении команды *Обработка матрицы – Максимум* максимальный элемент матрицы в окне редактора выделяется желтым цветом.

При выполнении команды *Обработка матрицы – Сортировка* появляется модальное окно с переключателями для выбора вида сортировки и полем ввода

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

Вариант 11

Написать Windows-приложение, работающее под управлением меню:

Файл
  Новый
  Открыть
  Сохранить
  Выход
Обработка
  Средняя зарплата
  Самый молодой
  Сортировка
 Об авторе

Для решения задачи создать класс *Rabotnik* содержащий следующие элементы: закрытые поля фамилия, название фирмы, зарплата за полугодие (массив), дата рождения, индексатор для доступа к элементам массива зарплат, метод для определения средней зарплаты, свойство для определения возраста.

При выполнении команды *Файл - Новый* появляется таблица *DataGridView* для ввода массива объектов класса *Rabotnik*, появляются кнопки *Сохранить* и *Обработка*, переключатели *Средняя зарплата, Самый молодой, Сортировка.* Пользователь получает возможность ввода и обработки информации.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла информация из него считывается в массив объектов и выводится в таблицу *DataGridView*, появляется кнопка *Обработка,* переключатели *Средняя зарплата, Самый молодой, Сортировка.*

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. После ввода имени информация сохраняется в соответствующем файле. То же самое происходит при нажатии кнопки *Сохранить.*

При выполнении команды *Обработка – Средняя зарплата* в отдельном окне выводится таблица со списком сотрудников заданной фирмы (название вводится с клавиатуры) и средней зарплатой за полугодие для каждого сотрудника. Аналогичные действия должны выполняться при нажатии кнопки *Обработка* и установленном переключателе *Средняя зарплата.*

При выполнении команды *Обработка – Самый молодой* в исходной таблице информация о самом молодом сотруднике подсвечивается красным цветом.

При выполнении команды *Обработка – Сортировка* в новом модальном окне выбирается переключатель: по возрасту, по среднему заработку. В таблицы *DataGridView* выводится исходный массив объектов и отсортированный.

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

Вариант 12

Написать Windows-приложение, работающее под управлением меню:

Файл
  Новый
  Открыть
  Сохранить
  Выход
Обработка
  Максимальная зарплата по каждому отделу
  Средняя
  Сортировка
 Об авторе

Для решения задачи создать класс *Sotrudnik* содержащий следующие элементы: закрытые поля фамилия, название отдела, зарплата за полугодие (массив), индексатор для доступа к элементам массива зарплат, метод для определения средней зарплаты; метод максимальной зарплаты за полугодие.

При выполнении команды *Файл - Новый* появляется таблица *DataGridView* для ввода массива объектов класса *Sotrudnik*, появляются кнопки *Сохранить* и *Обработка***,** переключатели *Максимальная зарплата, Средняя зарплата, Сортировка.* Пользователь получает возможность ввода и обработки информации.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла информация из него считывается в массив объектов и выводится в таблицу *DataGridView*, появляется кнопка *Обработка,* переключатели *Максимальная зарплата, Средняя зарплата, Сортировка*.

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. После ввода имени информация сохраняется в соответствующем файле. То же самое происходит при нажатии кнопки *Сохранить*.

При выполнении команды *Обработка- Максимальная зарплата* в отдельном окне в окне редактора *RichTextBox* выводится название

отдела и максимальная зарплата по отделу. Аналогичные действия должны выполняться при нажатии кнопки Обработка и установленном переключателе *Максимальная зарплата.*

При выполнении команды *Обработка – Средняя зарплата* в отдельном окне выводится таблица со списком сотрудников заданного отдела (название вводится с клавиатуры) и средней зарплатой за полугодие для каждого сотрудника в окне редактора *RichTextBox*. Аналогичные действия должны выполняться при нажатии кнопки *Обработка* и установленном переключателе *Средняя зарплата.*

При выполнении команды *Обработка – Сортировка* в новом модальном окне вывод списка всех сотрудников предприятия в порядке возрастания средней зарплаты осуществляется в таблице *DataGridView.*

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

Вариант 13

Написать Windows-приложение, работающее под управлением меню:

Файл
  Новый
  Открыть
  Сохранить
  Выход
Анализ текста
  Количество слов
  Количество предложений
  Выбор предложений
Об авторе

При выполнении команды *Файл - Новый* открывается окно *RichTextBox*, появляется кнопка *Сохранить* и панель с кнопкой *Анализ* и переключателями для выбора вида анализа текста:

Количество слов
Количество предложений
Выбор предложений

Пользователь получает возможность ввода текста.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла открывается окно *RichTextBox* с загруженным текстом, появляется кнопка *Сохранить* и панель с с кнопкой *Анализ* и переключателями для выбора вида анализа текста**.**

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. То же самое происходит при нажатии кнопки *Сохранить***.**

При выполнении команды *Анализ текста - Количество слов* определяется количество слов в тексте, состоящих не более чем из четырех букв. Команда дублируется кнопкой *Анализ***.**

При выполнении команды *Анализ текста - Количество предложений* определяется количество предложений в тексте, начинающиеся с тире, перед которыми могут находиться только пробельные символы заключенных в кавычки. Команда дублируется кнопкой *Анализ.*

При выполнении команды *Анализ текста - Выбор предложений* формируется массив предложений, содержащих двухзначные цифры, которые заменить словами (15- пятнадцать; 92 – девяносто два) присутствующих в тексте и выводится в окне редактора *RichTextBox*. Команда дублируется кнопкой *Анализ.*

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

Вариант 14

Написать Windows-приложение, работающее под управлением меню:

Файл
  Новый
  Открыть
  Сохранить
  Выход
Обработка массива
  Минимум
  Сумма…
  Преобразовать
Об авторе

Для решения задачи создать класс «Одномерный массив», в котором описать следующие элементы: закрытое поле – массив целых чисел, свойство для определения длины массива, индексатор для доступа к элементам поля-массива, конструктор с параметрами, перегруженные методы для вычисления суммы элементов, расположенных между первым и последним отрицательным элементом во всем массиве и для вычисления суммы в части массива, ограниченной начальным и конечным значениями индекса, передаваемых в метод в качестве параметров; метод поиска

минимального по модулю элемента массива; метод преобразования массива – в первой его половине располагались элементы, стоявшие в нечетных позициях, а во второй половине – элементы, стоявшие в четных позициях..

При выполнении команды *Файл - Новый* открывается окно *RichTextBox*, появляется кнопка *Сохранить***,** кнопка *Обработка* и раскрывающийся список для выбора вида обработки массива. Пользователь получает возможность ввода одномерного массива.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла открывается окно редактора *RichTextBox* с загруженным массивом, появляется кнопка *Обработка* и раскрывающийся список для выбора вида обработки массива**.**

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. После ввода имени массив сохраняется в соответствующем файле. То же самое происходит при нажатии кнопки *Сохранить*.

При выполнении команды *Обработка массива- Минимум* минимальный элемент подсвечивается красным цветом

При выполнении команды *Обработка массива – Сумма…* появляется окно с переключателями, определяющими, в какой части массива осуществляется вычисление суммы.

При выполнении команды *Обработка массива –Преобразовать* исходный массив и преобразованный выводятся в новом окне в таблицах *DataGridView*.

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

Вариант 15

Написать Windows-приложение, работающее под управлением меню:

Файл
- Новый
- Открыть
- Сохранить
- Выход

Обработка матрицы
- Поиск
- Максимум
- Перестановка

Об авторе

Для решения задачи создать класс «Матрица», в котором описать следующие элементы: закрытое поле – матрица целых чисел, индексатор для доступа к элементам поля-массива, конструктор с параметрами, метод поиска номера столбца, в котором находится самая длинная серия одинаковых элементов; перегруженные методы для максимального элемента, в части матрицы, ограниченной значениями индексов начальной строки и столбца и значениями индексов конечной строки и столбца, передаваемых в метод в качестве параметров и для поиска максимального элемента, расположенного выше побочной диагонали (с проверкой, является ли матрица квадратной); путем перестановки элементов квадратной (осуществить проверку) добиться того чтобы максимальный элемент каждой строки находился на главной диагонали (для первой строки в позиции (1,1), для второй – (2,2) ит.д).

При выполнении команды *Файл - Новый* открывается таблица *DataGridView* для ввода матрицы, появляется кнопка *Сохранить*, кнопка *Обработка* и панель с переключателями для выбора вида вычислений в матрице. Пользователь получает возможность ввода матрицы.

При выполнении команды *Файл – Открыть* открывается диалоговое окно открытия файла. После выбора файла открывается таблица *DataGridView* с загруженной матрицей, появляется кнопка *Обработка* и панель с переключателями для выбора вида вычислений в матрице.

При выполнении команды *Файл – Сохранить* открывается диалоговое окно сохранения файла. После ввода имени матрица сохраняется в соответствующем файле. То же самое происходит при нажатии кнопки *Сохранить***.**

При выполнении команды *Обработка матрицы- Поиск* **–** весь найденный столбец подсвечивается серым цветом или выдается сообщения, что такого столбца нет.

При выполнении команды *Обработка матрицы – Максимум…* появляется окно с переключателями, определяющими, где искать максимум. Максимальный элемент подсветить синим цветом.

При выполнении команды *Обработка матрицы – Перестановка* **–** исходная матрица и новая выводятся в новом окне в таблицах *DataGridView.*

При выполнении команды *Об авторе* открывается окно с информацией о разработчике программы с фотографией.

1.3 Вопросы для самоконтроля

1. Порядок создания Windows-приложения.
2. Шаблон Windows-приложения.
3. Свойства класса Control.
4. Основные методы класса Control.
5. Класс Form. Дополнительные свойства класса Form.
6. Класс Form. Методы класса Form.
7. Ввод данных с помощью элементов управления (Label и TextBox).
8. Ввод одномерного массива (элементы управления RichTextBox и DataGridView).
9. Ввод матрицы с помощью компонента DataGridView.
10. Вывод массивов с помощью компонента DataGridView.
11. Создание многооконных приложений, главного меню и вставка изображений.
12. Компонент RadioButton.
13. Компонент ListBox (список) и раскрывающийся список ComboBox.
14. Компонент ToolTip (всплывающая подсказка).
15. Компонент NumericUpDown.
16. Создание и использование диалоговых окон.

2 ИСПОЛЬЗОВАНИЕ ГРАФИЧЕСКИХ КЛАССОВ .NET

Цель: *получить навыки разработки программ, использующих графические классы Pen, Brush,Graphics, GraphicsPath, Image и др.*

2.1 Краткие теоретические сведения

Графический контекст представляет собой поверхность, обеспечивающую рисование на экране.

Графический объект класса *Graphics* управляет графическим контекстом.

Получить объект класса *Graphics* можно получить одним из следующих способов:

- путем вызова метода *CreateGraphics* объектов *Control* или *Form***;**
- из аргументов процедуры-обработчика события *Paint*;
- из изображения.

Первый способ можно проиллюстрировать оператором:

```
Graphics   gr = panel1.CreateGraphics();
```

Второй способ:

```
private void checkBox1_Paint(object sender, PaintEventArgs e)
        {
e.Graphics.DrawImage(Image.FromFile("f1.jpg"),0,0);
         }
```

Третий способ:

```
Image im =  Image.FromFile("IMG_0106.jpg");
Graphics g = Graphics.FromImage(im);

private void Form1_MouseDown( Object sender, MouseEventArgs e)
{
    // Получаем объект Graphics
          Graphics g = Graphics.FromHwnd(this.Handle);
// Теперь в месте щелчка мышью рисуем кружок диаметром 10 пикселов
g.DrawEllipse(new Pen(Color.Green),e.X, e.Y, 10, 10);
}
```

Класс *Pen*

Pen – класс пера, имеющий два основных атрибута: цвет и ширину.

Конструкторы класса *Pen*:

- `Pen(Color  color);`
- `Pen(Color color, float width).`

Например,

```
      Pen  pero = new Pen(Color.Red);
  Pen  pero1 = new Pen(Color.Red,5);
```

Для изображения линии используется метод класса *Graphics* *DrawLine* (перегружен 4 раза):

- *DrawLine(Pen p, Point p1,Point p2*) рисует линию пером *p* от точки *p1* до точки *p2*;
- *DrawLine(Pen p, float x1,float y1,float x2, float y2)* рисует линию пером *p* от точки (*x1,y1*) до точки (*x2,y2*);
- *DrawLine(Pen p, int x1,int y1,int x2, int y2)* рисует линию пером p от точки (*x1,y1*) до точки (*x2,y2*).

Brush – абстрактный класс. Для создания кисти используются классы, производные от него:

- *HatchBrush* – прямоугольная кисть для заполнения области узором;
- *LinearGradientBrush* – заполнение области постепенным смешением одного цвета с другим. Линейные градиенты можно задать двумя цветами, углом градиента и шириной прямоугольника, (или двумя точками);
- *SolidBrush* – заполнение области одним цветом;
- *TextureBrush* – заполнение области повторением заданного изображения (*Image*).

В пространстве *System.Drawing* есть коллекция готовых кистей *Brushes*.

В классе *Graphics* определены следующие нестатические методы:

- *TranslateTransform(float dx, float dy)* смещает начало координат на *dx*;
- *RotateTransform(float angle)* – поворачивает систему координат на заданный в качестве параметра угол;
- *ResetTransform( )* – отменяет преобразование координат объекта *Graphics*.

Для изображения строки текста можно использовать метод класса Graphics:

DrawString(string s, Font f, Brush b, float x, float y)

Вывести на экран векторное или растровое изображение позволяет класс *System.Drawing.Image*.

Всякая компьютерная картинка относится к одной из этих двух категорий, поэтому у абстрактного класса *Image* есть два конкретных потомка – классы *Metafile* и *Bitmap*.

Чтобы получить растровое изображение, нужно создать объект *Bitmap* из какого-нибудь графического файла.

```
Image image = new Bitmap("myPhoto.bmp");
```

Теперь изображение можно вывести на поверхность рисования.

```
Graphics g = this.CreateGraphics();
g.DrawImage(image, new Rectangle(10, 10, 200, 100));
```

Класс *Image* определяет множество свойств и методов, которые можно использовать для настройки параметров выводимого изображения. К примеру, при помощи свойств *Width, Height* и *Size* можно получить или установить размеры изображения. Кроме того, в пространстве имен *System.Drawing.Imaging* определено множество типов для проведения сложных преобразований изображений.

Наиболее важные члены класса *Image* представлены в таблице 2.1. Многие из этих членов являются статическими, а некоторые – абстрактными.

Таблица 2.1 Члены класса Image

Член	Назначение
FromFile()	Этот статический метод предназначен для создания объекта *Image* из файла
FromHbitmap()	Создает объект *Bitmap* на основе идентификатора окна (Window handle)
FromStream()	Позволяет создать объект *Image*, используя в качестве источника поток данных
Height *Width* *Size* *Physical Dimensions* *Horizontal* *Dimensions* *Vertical Resolution*	Все эти свойства предназначены для работы с размерами (измерениями) изображения
GetBounds()	Возвращает прямоугольник, представляющий текущую область, занятую изображением
Save()	Позволяет сохранить изображение в файл

Класс *Image* является абстрактным, и создавать объекты этого класса нельзя. Обычно объявленные переменные *Image* присваиваются объектам класса *Bitmap*. Кроме того, можно создавать объекты класса *Bitmap* напрямую. Например, предположим, что необходимо вывести на форму три изображения. Можем объявить три переменные *Image*, а затем использовать для каждой из них объекты *Bitmap*.

В классе *Graphics* имеется тридцать перегруженных методов *DrawImage()*. Все они выводят изображение, но с какими-то

преобразованиями – растяжение, сдвиг, скашивание, зеркальное отражение.

- *DrawImage(Image im, int x, int y)* – выводит изображение, совмещая левый верхний угол изображения с точкой (*x, y*).

Например,

```
Graphics  g = panel1.CreateGraphics();
                    g.DrawImage(picture, 0, 0);
```

- *DrawImage(Image im, int x, int y, int width, int height)* – выводит изображение размером *width u height***.**

Например,

```
  g.DrawImage(picture, 0, 0,300,200);
picture.RotateFlip(RotateFlipType.Rotate180FlipX);
  g.DrawImage(picture, 0, 0,300,200);
```

Класс *Bitmap* позволяет выводить изображения, которые хранятся в файлах самого разного формата. Например:

// Тип Bitmap поддерживает все распространенные форматы!

```
Bitmap myBMP = new Bitmap("CoffeeCup.bmp");
Bitmap myGIF = new Bitmap("Candy.gif");
Bitmap ntyJPEG = new Bitmap("Clock.jpg");
Bitmap myPNG = new Bitmap("Speakers.png");
```

// Выводим изображения при помощи *Graphics.DrawImage()*

```
g.DrawImage(myBmp, 10, 10);
g.DrawImage(myGIF, 220, 10);
g.DrawImage(myJPEG, 280, 10);
g.DrawImage(myPNG, 150, 200);
```

Для загрузки изображения из графического файла создается объект класса *Image***.**

Например,

```
Image  picture =  Image.FromFile("IMG_0106.jpg");
```

Способ уменьшения мерцания – использовать *двойную буферизацию вывода* – когда изображение сначала готовится "за кадром", а затем переносится на экран, устраняя мерцание при анимации.

Пример 2.1 Написать программу для рисования линий и многоугольников (рис. 2.1 и 2.2).

Предусмотрим возможность отображения линий и многоугольников при перемещении указателя мыши. При двойном щелчке рисование прекращается.

В класс *Form1* добавим поле

int select = 0;

После щелчка мыши *select = 1,* после двойного щелчка *select = 2.*

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Collections;
namespace WindowsFormsApplication1
{public partial class Form1 : Form
    {ArrayList points = new ArrayList();
        Pen p = new Pen(Color.DarkGreen);
        int   select = 0;
public Form1()
        {InitializeComponent()}
private void panel1_MouseDown(object sender, MouseEventArgs e)
        {
            if (select != 2)
            {
                select = 1;
points.Add(new Point(e.X, e.Y));
panel1.Invalidate();
            }}
private void button1_Click(object sender, EventArgs e)
        {
            select = 0;
            points.Clear();
            panel1.Invalidate();}
private void panel1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
if (points.Count > 1)
            {
 Point[] arrayofpoints =
(Point[])points.ToArray(points[0].GetType());
if (radioButton1.Checked)
                    g.DrawLines(p, arrayofpoints);
                else
                    g.DrawPolygon(p, arrayofpoints);
if (select == 0) points.RemoveAt(points.Count - 1);
select = 0;
}}
private void panel1_MouseDoubleClick(object sender,
MouseEventArgs e)
        {select = 2;        }
private void panel1_MouseMove(object sender, MouseEventArgs e)
        {
            if (select != 2)
            {Point pp = new Point(e.X, e.Y);
                if (points.Count > 0 && select != 2)
                    points.Add(pp);
                panel1.Invalidate();
            }}}}
```

Рисунок 2.1 – Листинг программы

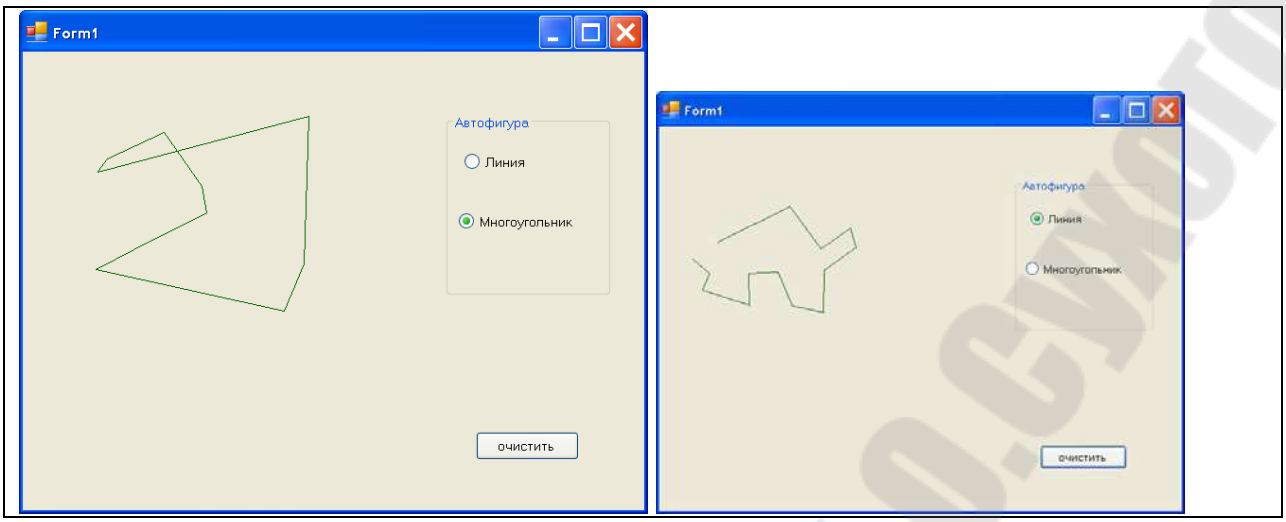


Рисунок 2.1 – Результат работы программы 2.1

Пример 2.2 Разработать программу, которая демонстрирует перемещения мяча по ровной поверхности (рис. 2.3, 2.4).

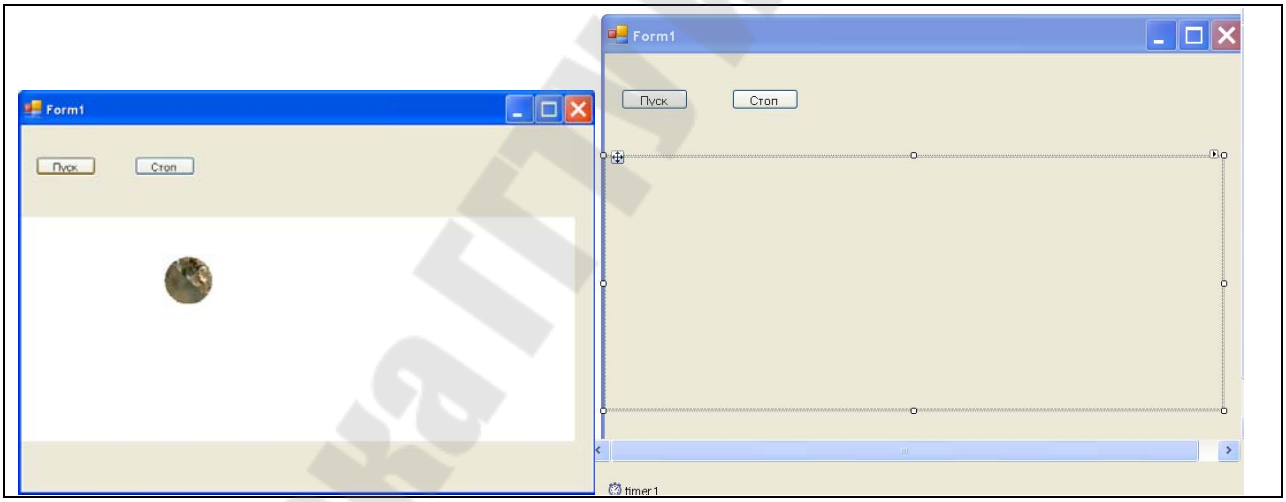


Рисунок 2.3 – Результат работы программы 2.2

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{public partial class Form1 : Form
    {Graphics gr;
        int i = 0; int x = 80;
        float dx = 10;int x1 = 15;
public Form1()
        {InitializeComponent();
            gr = panel1.CreateGraphics();
          gr.TranslateTransform(80, 80);
             timer1.Tick += new
EventHandler(TimerEventProcessor);}
private void button1_Click(object sender, EventArgs e)
        {timer1.Start();}
        private void button2_Click(object sender, EventArgs e)
        {timer1.Stop();}
private void panel1_Paint(object sender, PaintEventArgs e)
        {gr.Clear(Color.White);}
public void TimerEventProcessor(Object myObject, EventArgs
myEventArgs)
        {SolidBrush w = new SolidBrush(Color.White);
TextureBrush br1=new TextureBrush(Image.FromFile("1462.jpg"));
            gr.FillEllipse(w,-31,-31,62,62);
            if (i==8)
            {gr.TranslateTransform(x1,0); i=0;}
            if (i==2)
            {gr.TranslateTransform(0,-x1);}
           if (i==1)
            {gr.TranslateTransform(dx,-dx);}
            if (i==3)
            {gr.TranslateTransform(-dx,-dx);}
            if (i==5)
            {gr.TranslateTransform(-dx,dx);}
            if (i==6)
            {gr.TranslateTransform(0,x1);}
            if (i==4)
            {gr.TranslateTransform(-x1,0);}
            if (i==7)
            {gr.TranslateTransform(dx,dx);}
gr.FillEllipse(br1,-30,-30,60,60);
            gr.RotateTransform(45);x=x+15;
            if(x>panel1.Width-5)
            {dx=-dx; x1=-x1; x=40;}
            i=i+1;}}}
```

Рисунок 2.4 – Листинг программы 2.2

2.2 Варианты заданий

Вариант 1

Разработать приложение, которое строит график выбранной функции на заданном интервале.

Выбор функции осуществлять посредством меню из нескольких заранее определенных вариантов.

Для выбора типа линии использовать переключатели RadioButton.

Для выбора цвета графика использовать listBox.

График должен строиться в новом окне. В окне должно отображаться название графика, оси должны иметь разметку.

Точки пересечения с осью абсцисс (если они есть на заданном интервале) пометить квадратными маркерами.

Вариант 2

Разработать простейший редактор для построения графической схемы линейного алгоритма.

Выбор элементов схемы осуществлять с помощью RadioButton.

Вариант 3

Разработать приложение, которое открывает текстовый файл, содержащий табличную функциональную зависимость (в виде двух столбцов), выводит таблицу значений функции, строит график зависимости по выбору пользователя либо в виде столбиковой диаграммы, либо в виде кривой.

При этом максимальное значение функции отмечается либо цветом столбика, либо треугольным маркером красного цвета на графике.

Вариант 4

Разработать простейший графический редактор для построения рисунка из различных фигур (линий, прямоугольников, дуг окружностей, многоугольников). Выбор фигуры осуществлять щелчком мыши по изображению соответствующей фигуры на специальной панели. Выбор цвета осуществлять с помощью стандартного диалога.

Вариант 5

Разработать программу, которая позволяет строить график заданной функции, определяющей траекторию движения материальной точки. Траектория задается аналитически (видом функции $F(x)$) и выбирается из трех предложенных вариантов с помощью меню.

При нажатии на соответствующую кнопку программа должна демонстрировать движение материальной точки (представленной в виде звездочки) по изображенной траектории.

Вариант 6

Создать текстовые файлы *ЗИС-21.txt* и *ЗИС-22.txt* с информацией об успеваемости студентов по курсу «Объектно-ориентированное программирование» (Фамилия студента, оценка). Разработать приложение, позволяющее отобразить информацию по выбору пользователя в виде столбиковой диаграммы с подписями фамилий студентов или в виде круговой диаграммы, отражающей процентное соотношение количества двоечников, отличников и остальных студентов в группе. Предусмотреть возможность выбора цвета столбиков с помощью стандартного диалога.

Вариант 7

Разработать приложение, позволяющее строить график сразу нескольких функций (число функций не больше 4) на одном графическом поле. Предусмотреть вывод шкалы по осям координат и отображение сетки с заданным числом линий разбиения по осям. Функции могут быть заданы аналитически (выбираться из предложенных вариантов с помощью компонента *CheckBox*) или таблично (данные хранятся в текстовых файлах).

Вариант 8

Разработать приложение, позволяющее строить график исходной зависимости, заданной таблично и записанной в текстовый файл, в виде отдельных точек (маркеров) и график аппроксимирующей функции вида Ax^2+Bx+C, коэффициенты которой получены методом наименьших квадратов.

Вариант 9

Разработать приложение, которое строит график выбранной функции на заданном интервале.

Выбор функции осуществлять с помощью переключателя *RadioButton* из нескольких заранее определенных вариантов.

Для выбора типа линии использовать *ComboBox*.

Для выбора цвета графика использовать стандартный диалог.

График должен строиться в том же окне на отдельной панели. Если график пересекает оси координат, оси с разметкой отображать.

Вариант 10

Разработать приложение, которое открывает текстовый файл, содержащий в первой строке начальное значение аргумента функции и его конечное значение, а во второй строке значения функции (10<количество значений<50), выводит таблицу значений функции в отдельном окне, строит график зависимости. При нажатии на соответствующую кнопку и установке границ интервала часть графика функции, где значения функции попадают в заданный интервал (a<f(x)<b), окрашивается в красный цвет.

Вариант 11

Разработать программу, которая позволяет строить график заданной функции, определяющей траекторию движения материальной точки. Траектория задается таблично (значения аргумента и функции хранятся в текстовом файле).

При нажатии на соответствующую кнопку программа должна демонстрировать движение материальной точки по изображенной траектории. Вид материальной точки выбирается из трех заданных вариантов: звезда, треугольник, круг.

Вариант 12

Разработать программу, которая демонстрирует перемещение мяча по ровной поверхности. Цвет и тип заливки мяча определять с помощью меню.

Вариант 13

Разработать программу отображения движущегося автомобиля (паровоза) с крутящимися колесами (путем изображения “спиц” на колесах).

Цвет и тип заливки блоков, из которых состоит автомобиль(паровоз), выбрать с помощью меню

Вариант 14

Разработать приложение, позволяющее строить график исходной зависимости (заданной таблично и записанной в текстовый файл) в виде столбиковой диаграммы, и график аппроксимирующей функции вида $Ax^2+ B$, коэффициенты которой получены методом наименьших квадратов.

Вариант 15

Создать текстовые файлы *ЗИС-21.txt* и *ЗИС -22.txt* с информацией об успеваемости слушателей по курсу «Операционные системы» (Фамилия студента, оценка). Создать также файлы с фотографиями студентов: Янчев.jpg, Капачев.jpg ит.д.

Разработать приложение, которое строит гистограмму (столбиковую диаграмму) оценок студентов, вместо подписей фамилий по оси *OX*. Данные о двоечниках выводятся красным цветом.

По кнопке «Отчислить» двоечники удаляются из файла, на диаграмме это иллюстрируется анимацией (например, постепенным уменьшением соответствующих столбиков до полного исчезновения).

2.3 Вопросы для самоконтроля

1. Обзор графических классов и структур.
2. Способы создания графических объектов.
3. Класс Pen. Свойства класса Pen.
4. Рисование линий.
5. Обновление области рисования.
6. Рисование геометрических фигур с помощью методов класса Graphics.
7. Абстрактный класс Brush.
8. Отображение сложных фигур. Методы класса GraphicsPath.
9. Преобразование системы координат.
10. Изображение строки текста.
11. Вывод изображений. Члены класса Image.
12. Класс *Bitmap.*
13. Поверхность в памяти.
14. Вывод изображений и двойная буферизация.
15. Перетаскивание, проверка попадания в область, занимаемую изображением. Элемент управления PictureBox.
16. Самодельные элементы управления
17. Наследование от UserControl.
18. Наследование от *Control*

3 РАЗРАБОТКА ИГР С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ

Цель: *получить навыки разработки программ-игр, использующих формы, элементы управления и принципы событийно-управляемого программирования.*

3.1 Краткие теоретические сведения

Рассмотрим классический пример – игра в бильярд, т.е. на экране стол, на столе шары, шары движутся, отражаются от стенок, сталкиваются, останавливаются, падают в лузы. Начнем с того, что нарисуем прямоугольный стол, пока без луз. На столе несколько шаров, шары движутся по столу и отражаются от стенок. Между собой шары еще не взаимодействуют, каждый шар ведет себя, как будто он на столе один.

Хоть программа простая, без проектирования не обойтись. За основу возьмем схему «Модель – Отображение». В модели сразу видны два класса: *Шар* и *Стол***,** причем *Стол* владеет коллекцией шаров. Кроме коллекции шаров, у *Стола* есть *ширина* и *высота*. Состояние шара включает его местоположение на столе и скорость. Радиусы всех шаров одинаковы, поэтому в состояние шара радиус не входит.

Как шары будут двигаться? Маленькими шажками.

Дело в том, что каждая Windows-программа представляет из себя бесконечный цикл, внутри которого все и происходит. Этот цикл скрыт в методе *Application.Run*, поэтому простым глазом не виден, но он есть. Когда цикл разрывается, например, при закрытии главной формы, программа завершается. На каждом витке цикла происходит обработка всех возникших к этому времени событий. Именно поэтому обработка каждого события должна быть короткой, если она будет долгой, то выполнение одного витка задержится и все в программе замрет на это время.

Значит, на каждом витке цикла все шары должны сделать по маленькому шажку, тогда возникнет ощущение непрерывного движения? Не совсем. Главный цикл от нас спрятан, к тому же витки выполняются слишком быстро, если за один виток смещать шар хотя бы на один пиксель, он пулей вылетит за экран.

Выход есть, он в компоненте *Timer*. Включим экземпляр таймера в программу, и он послужит источником периодических событий, которые можно обрабатывать. При помощи свойств таймера можно установить разумный интервал между его событиями, например, 50 миллисекунд и это то, что нам нужно.

Теперь можно представить код классов *Шар* и *Стол*. Сначала Шар.

```
using System;
namespace Snooker {
    class Ball {
     internal const int R = 20, D = R + R;
    internal int X { set; get; }
    internal int Y { set; get; }
    internal int Vx { set; get; }
    internal int Vy { set; get; }
    internal Table Table { set; get; }
    internal Ball(int x, int y, int vx, int vy, Table table) {
        X = x;
        Y = y;
        Vx = vx;
        Vy = vy;
        Table = table; }
 internal void Step() { int x = X + Vx, y = Y + Vy;
  // Изменение скорости в результате отражения от борта стола
    if (x > Table.Width - R || x < R) { Vx = -Vx; }
  if (y > Table.Height - R || y < R) { Vy = -Vy;}
   // Изменение координат
    X += Vx; Y += Vy;   }      } }
```

Немного пояснений: *Шар* имеет четыре свойства: *X* и *Y* – координаты шара относительно стола, *Vx* и *Vy* – проекции скорости шара на оси координат. *Vx* – это количество пикселей, на которое переместится шар за один шаг в горизонтальном направлении, а *Vy* – в вертикальном.

В объект шара добавлена ссылка на стол:

```
    internal Table Table { set; get; }
```

Код единственного метода *Step()* состоит из двух частей. В первой проверяется, не пора ли шару отразиться от борта, а во второй делается тот самый маленький шаг.

Код класса Table

```
using System;
using System.Collections.Generic;
 namespace Snooker {
  class Table {
   internal int Width { set; get; }
   internal int Height { set; get; }
   internal List<Ball> Balls { set; get; }
   internal Table(int width, int height) {
       Width = width;
       Height = height;
       Balls = new List<Ball>();   }
  internal void Step() { foreach (Ball ball in Balls) {ball.Step();}     }   }   }
```

У *Стола* есть ширина, высота и коллекция шаров. Сразу после создания стола коллекция пуста, шары надо добавлять на стол позже. Метод *Step()* просто вызывает методы Step() всех шаров коллекции.

За отображение по традиции отвечает форма. На форме находится единственный компонент – *Timer* по имени *timer*. В окне свойств видно, что интервал между событиями установлен в 20 мсек (рис. 3.1).

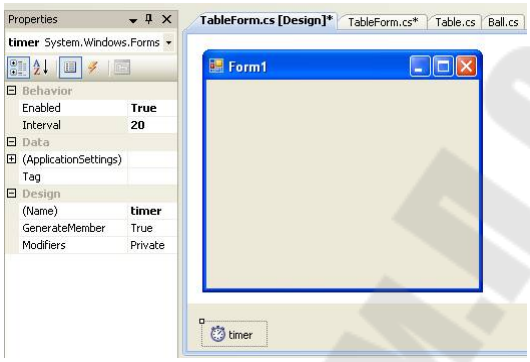


Рисунок 3.1 – Форма с timer

На рисунке 3.2 приведен полный текст файла TableForm.cs.

```
    using System;
 using System.Drawing;
 using System.Windows.Forms;
 namespace Snooker {
   public partial class TableForm : Form
    {
// Это часть формы, которую будет занимать стол
readonly Rectangle tableRect = new Rectangle(40, 40, 600, 400);
    Table table;
// Заполняет стол шарами, движущимися с разной скоростью
  public TableForm() {
   InitializeComponent();
     table = new Table(tableRect.Width, tableRect.Height);
       for (int i = 1; i < 10; ++i) {
table.Balls.Add(new Ball(300, 200, i, i, table)); } }
 private void timer_Tick(object sender, EventArgs e)
        { table.Step(); DrawTable(); }
  private void DrawTable()
{ Graphics g = this.CreateGraphics();
  g.FillRectangle(Brushes.DarkGreen, tableRect);
// Зеленое сукно
foreach (Ball b in table.Balls)
{ g.FillEllipse(Brushes.Ivory,
// Шары из слоновой кости
tableRect.Left + b.X - Ball.R,
tableRect.Top + b.Y - Ball.R,
Ball.D, Ball.D); }}}     }
```

Рисунок 3.2 – Текст файла TableForm.cs

В конструкторе формы создаем стол и наполняем его шарами. В обработчике события от таймера делаем один шаг из жизни стола и рисуем стол в его текущем состоянии. Рисованием занимается метод формы *DrawTable()*. Рисуем в «легкомысленном» стиле, получая объект Graphics методом формы *CreateGraphics()*. В данном случае это приемлемо, т.к. изображение все равно обновляется много раз в секунду.

Мы остановились на том, что шары неприятно помаргивали, даже когда не двигались. Это происходило оттого, что на экране сначала появлялся зеленый фон, а потом белые шары и так много раз в секунду. Область экрана под шаром становилась сначала зеленой, потом белой, потом опять зеленой, потом опять белой….

Чтобы помочь беде, нарисуем стол с шарами сначала в памяти, т.е. на поверхности пустого изображения нужного размера. После того, как рисование в памяти будет закончено, выведем изображение на поверхность формы методом *DrawImageUnscaled()*.

Это усовершенствование должно быть сделано в методе формы *DrawTable*(), который первоначально был таким.

```
  private void DrawTable() {
        Graphics g = this.CreateGraphics();
// Зеленое сукно.
g.FillRectangle(Brushes.DarkGreen, tableRect);
         // Шары из слоновой кости
         foreach (Ball b in table.Balls) {
g.FillEllipse(Brushes.Ivory,
tableRect.Left + b.X - Ball.R,
tableRect.Top + b.Y - Ball.R,
Ball.D, Ball.D);
         }
      }
```

А теперь станет таким.

```
private void DrawTable() {
Bitmap buffer = new Bitmap(table.Width, table.Height);
Graphics bg = Graphics.FromImage(buffer);
// Зеленое сукно.
bg.FillRectangle(Brushes.DarkGreen, 0, 0,
table.Width, table.Height);
 // Шары из слоновой кости
foreach (Ball b in table.Balls) {
bg.FillEllipse(Brushes.Ivory,
b.X - Ball.R, b.Y - Ball.R, Ball.D, Ball.D);
         }
// Переносим содержимое буфера на экран.
Graphics g = this.CreateGraphics();
g.DrawImageUnscaled(buffer, tableRect.Location); }
```

Можно пойти дальше и в качестве буфера использовать не пустое изображение, а рисунок пустого стола и не понадобится рисовать сукно.

Запрограммируем столкновение двух шаров. С точки зрения программы, если два шара сближаются (расстояние между их центрами меньше или равно диаметру шара), их скорости каким-то образом должны измениться.

Изменение скоростей у нас происходило в методе *Ball.Step()*. Теперь он немного усложнится.

```
internal void Step() {
// Изменение скорости в результате соударения
Ball other = WhatToStrike();
if (other != null) {
            Strike(other);
          }
// Изменение скорости в результате отражения от борта стола
float x = X + Vx, y = Y + Vy;
if (x > Table.Width - R || x < R) {
            Vx = -Vx;
          }
if (y > Table.Height - R || y < R) {
            Vy = -Vy;
          }
          // Изменение координат
         X += Vx;
        Y += Vy;
      }
```

Метод *WhatToStrike()* должен находить шар, вплотную приблизившийся к шару *this*. Если такого шара нет, он возвращает *null*. Коллекция всех шаров доступна через свойство *Table.Balls*.

Метод *Strike()* изменяет скорости двух шаров – *this* и other. Как именно меняются скорости, мы разберемся позже, а пока (временно) пусть будет простой обмен, скорость первого шара станет скоростью второго и наоборот.

```
       private void Strike(Ball b2) {
float t = Vx; Vx = b2.Vx; b2.Vx = t;
t = Vy; Vy = b2.Vy; b2.Vy = t;
      }
```

Такие временные методы называются заглушками. Они позволяют быстро увидеть работу программы, пусть и не совсем правильную.

В процессе написания программы нам придется много экспериментировать с различным положением и скоростями шаров, поэтому будем добавлять шары на стол при помощи мыши.

Обработаем события *MouseDown* и *MouseUp* для формы.

```
Point startPoint;
   private void TableForm_MouseDown(object sender,
MouseEventArgs e) {
startPoint = e.Location;
   }

   private void TableForm_MouseUp(object sender, MouseEventArgs
e) {
table.Balls.Add(new Ball(e.X - tableRect.X, e.Y - tableRect.Y,
(e.X - startPoint.X) / 5, (e.Y - startPoint.Y) / 5, table));
   }
```

Новый шар появляется на столе в момент отпускания кнопки мыши. Его скорость тем больше, чем дольше мы протянули мышь с нажатой кнопкой. Шар покатится в направлении движения мыши (рис. 3.3).

Рисунок 3.3 – Добавление шаров

Сейчас мы напишем программу, которая рассчитает скорости двух шаров после соударения.

Сделаем два предположения. Первое, шары абсолютно скользкие, т.е. ударяются без трения. Второе, удар абсолютно упругий, т.е. никакая часть энергии движения шаров не превращается в тепло. Предположения хоть и не реальные, но необходимые, без них нам с расчетом не справиться.

Сначала рассмотрим случай, когда центры шаров находятся на прямой, параллельной оси Ox, второй шар неподвижен, а скорость первого шара направлена точно к центру второго шара (рис 3.4).

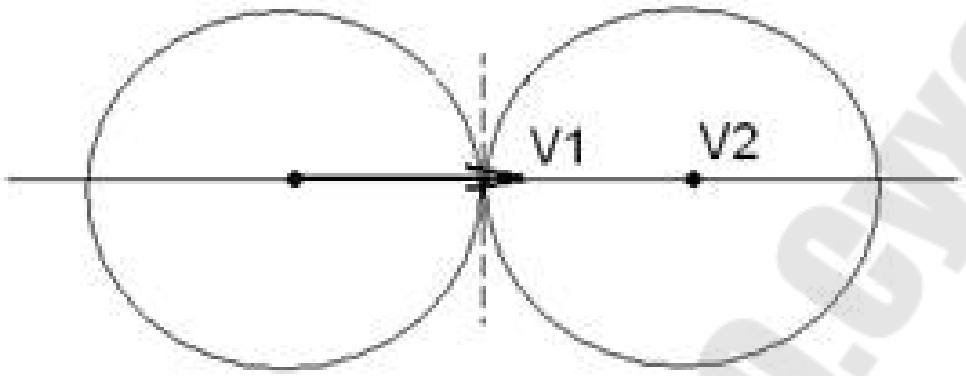


Рисунок 3.4 – Центры шаров находятся на прямой, параллельной оси Ox

Пусть V_1 и V_2 – скорости шаров до соударения ($V_2 = 0$), U_1 и U_2 – скорости шаров после соударения, m – масса шара.

По закону сохранения импульса:

$$mV_1 + mV_2 = mU_1 + mU_2 ;$$

По закону сохранения энергии:

$$mV_1^2 / 2 + mV_2^2 / 2 = mU_1^2 / 2 + mU_2^2 / 2 .$$

Поскольку $V_2 = 0$, а массы шаров одинаковы, система уравнений сильно упростится:

$$V_1 = U_1 + U_2 ;$$
$$V_1^2 = U_1^2 + U_2^2 .$$

Возведем первое уравнение в квадрат $V_1^2 = U_1^2 + 2*U_1*U_2 + U_2^2$ и сравним со вторым.

Получается, что

$$2*U_1*U_2 = 0.$$

Т.е. после соударения либо скорость первого шара, либо скорость второго шара равна нулю. Второе предположение абсурдно, оно означает, что первый шар прошел через второй, как сквозь воздух. Значит, $U_1 = 0$, а $U_2 = V_1$ (это следует из первого уравнения).

Теперь шары расположены так же, но скорость движущегося шара направлена под углом к прямой, соединяющей центры шаров (рис. 3.5).

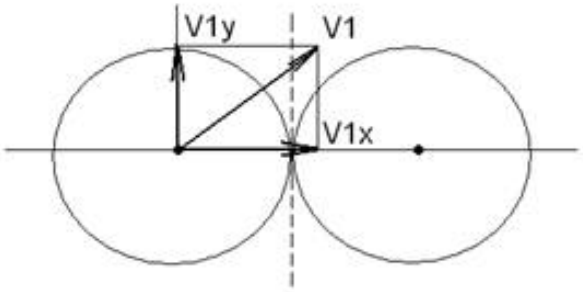


Рисунок 3.5 –Скорость движущегося шара направлена под углом к прямой, соединяющей центры шаро

Скорость шара V представлена в программе двумя составляющими: V_x – проекцией на ось Ox и V_y – проекцией на ось Oy.

Все, что ранее было сказано в отношении скоростей до и после соударения, теперь справедливо только в отношении их проекций на ось Ox, то есть

$$U_{1x} = 0;$$
$$U_{2x} = V_{1x}.$$

Что касается проекций скоростей на ось Oy, они не меняются, т.к. в этом направлении шары не взаимодействуют (трения же нет), т.е.

$$U_{1y} = V_{1y};$$
$$U_{2y} = 0.$$

Если второй шар не стоит на месте, а движется со скоростью V2, это легко учесть заменой системы отсчета.

1. Переходим в систему отсчета, в которой второй шар неподвижен. Для этого вычтем скорость V2 из скорости каждого шара.
2. Выполняем обмен скоростей по формулам
   $U_{1x} = 0$; $U_{1y} = V_{1y} - V_{2y}$;
   $U_{2x} = V_{1x} - V_{2x}$; $U_{2y} = 0$.
3. Возвращаемся в первоначальную систему отсчета. Для этого добавим скорость V2 к скорости каждого шара.
   $U_{1x} = V_{2x}$; $U_{1y} = V_{1y}$;
   $U_{2x} = V_{1x}$; $U_{2y} = V_{2y}$.

Получается, что в направлении Ox шары обмениваются скоростями, а в направлении Oy свои скорости сохраняют.

Итак, все довольно просто, если центры шаров находятся на прямой, параллельной оси Ox, но как же быть в общем случае, когда эта прямая составляет с осью Ox произвольный угол α?

1. Переходим в систему координат, в которой линия центров параллельна Ox, иными словами делаем поворот обоих шаров на угол α вокруг начала координат.
2. Выполняем обмен скоростей по формулам
   $U_{1x} = V_{2x}$; $U_{1y} = V_{1y}$;
   $U_{2x} = V_{1x}$; $U_{2y} = V_{2y}$.
3. Возвращаемся в первоначальную систему координат, т.е. делаем поворот шаров на угол -α.

Как сделать поворот. Посмотрите на рисунок (рис. 3.6). Можно сказать, что точка R есть векторная сумма точек A и *B*, здесь полная аналогия со скоростью.

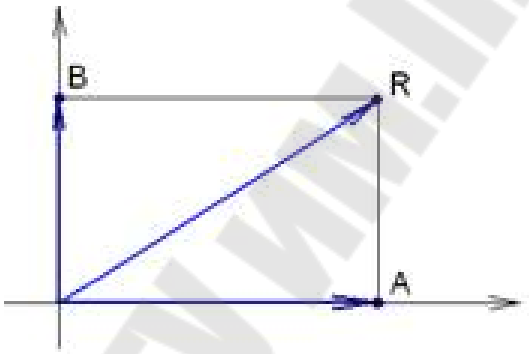


Рисунок 3.6

Если координаты точки *R* – (*x, y*), то точки *A* и *B* имеют следующие координаты
A = (*x, 0*), *B* = (*0, y*).

Вместо того, чтобы поворачивать на угол α точку *R*, повернем точки *A* и B (рис. 3.7).

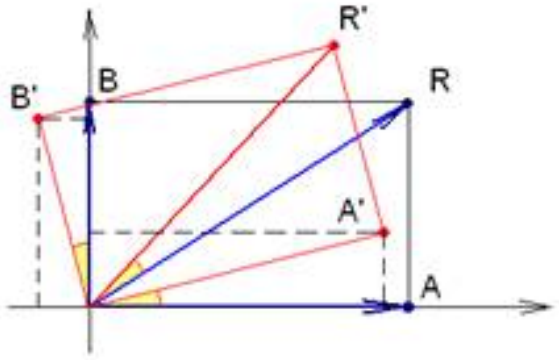


Рисунок 3.7

Координаты точек *A'* и *B'* теперь равны

$A' = (x * cos(\alpha), x * sin(\alpha))$

$B' = (– y * sin(\alpha), y * cos(\alpha))$

Точка R' – сумма точек A' и B', т.е.

$R' = (x * cos(\alpha) – y * sin(\alpha), x * sin(\alpha) + y * cos(\alpha))$

Вот мы и получили формулы для определения координат повернутой точки R:

$x' = x * cos(\alpha) – y * sin(\alpha)$;

$y' = x * sin(\alpha) + y * cos(\alpha)$.

Здесь x и y – координаты точки до поворота, x' и y' – координаты точки после поворота.

Еще нужно уметь делать поворот на –α. Это совсем просто, если понимать, что функция косинуса симметрична относительно оси Oy, т.е. cos(–α) = cos(α), а функция синуса центрально симметрична относительно начала координат, т.е. sin(–α) = –sin(α).

Формулы для поворота на –α будут такими:

$x' = x * cos(\alpha) + y * sin(\alpha)$;

$y' = –x * sin(\alpha) + y * cos(\alpha)$.

Наконец можно написать метод для расчета скоростей после соударения двух шаров.

```
  private void Strike(Ball b2)
    {
       Ball b1 = this;
// Определение угла между линией центров шаров и осью Ox.
// Нужен не сам угол, а его синус и косинус.
float dx = b1.X - b2.X,
dy = b1.Y - b2.Y,
dd = (float)Math.Sqrt(dx * dx + dy * dy);
float sinA = dy / dd,
cosA = dx / dd;
// Поворот на угол A.
float b1Vx =  b1.Vx * cosA + b1.Vy * sinA;
float b1Vy = -b1.Vx * sinA + b1.Vy * cosA;
float b2Vx =  b2.Vx * cosA + b2.Vy * sinA;
float b2Vy = -b2.Vx * sinA + b2.Vy * cosA;
// Обмен скоростями Vx между шарами b1 и b2.
 loat t = b1Vx; b1Vx = b2Vx; b2Vx = t;
// Поворот на угол -A.
b1.Vx = b1Vx * cosA - b1Vy * sinA;
b1.Vy = b1Vx * sinA + b1Vy * cosA;
b2.Vx = b2Vx * cosA - b2Vy * sinA;
b2.Vy = b2Vx * sinA + b2Vy * cosA;
    }
```

Если некоторые шары будут слипаться, следует поработать над методом *WhatToStrike()* – поиск шара, с которым предстоит столкнуться шару *this*.

3.3 Варианты заданий

Вариант 1 Игра «Пятнадцать»

Правила игры. На игровом поле размером 16 клеток (4 на 4) случайным образом размещается 15 фишек с номерами от 1 до 15. Таким образом, шестнадцатая клетка остается пустой. Игрок может, нажатием на смежную с пустой клеткой по вертикали или горизонтали фишку, переместить ее на пустую клетку. Соответственно, ранее занимаемая этой фишкой клетка становится пустой. Цель игры – путем таких перемещений выстроить фишки по номерам, начиная с верхнего левого угла. Если игроку это удалось, игра завершается сообщением «победа». Если получился вырожденный случай (фишки 14 и 15 оказались поменяны местами, то есть выиграть невозможно), то игра завершается сообщением «Не повезло».

Обязательно наличие счетчика ходов и возможности отмены последних сделанных ходов, вплоть до первого хода.

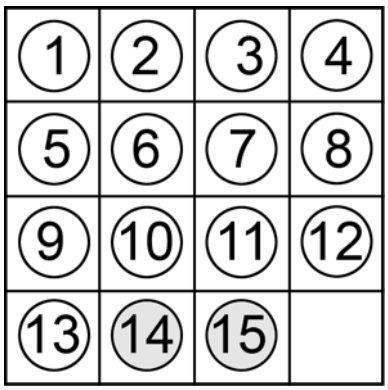
Также предусмотреть вариант игры на время, по истечении которого игра завершается проигрышем «Время вышло».

Подсказки по выполнению.

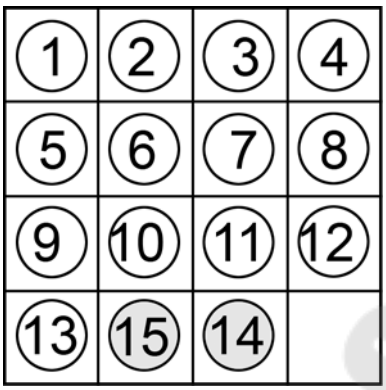
Массив фишек можно задать от 0 до 15, где число 0 будет означать пустую клетку.

Алгоритм заполнения поля фишками в случайном порядке: массив фишек задается связным списком, из которого случайным образом выбирается элемент, который размещается на поле (последовательно) и одновременно удаляется из списка. Процесс зацикливается до тех пор, пока в списке не останется 0 элементов.

Процесс проверки на смежность: если у выделенной клетки с пустой клеткой один индекс совпадает, а второй различается на единицу, значит они смежны. Например, клетки (4,2) и (4,3). Если пользоваться одномерной нумерацией, то тогда первый индекс получается из номера элемента деленного нацело на 4 и +1, а второй индекс – остаток от деления на 4. Например, элемент, стоящий на 9 месте в одномерном массиве, в двумерном будет располагаться в клетке 9 div 4+1 = 3, 9 mod 4 = 1, то есть в клетке (3,1).



Финальное расположение



Вырожденный случай

Вариант 2 Игра «Реверси»

Правила игры. Играют двое игроков на поле размером 8 на 8 клеток. Один играет фишками белого цвета, другой – черного. Игроки выставляют свои фишки по очереди из начальной позиции по следующим правилам.

1. Если между выставленной игроком фишкой и любой фишкой его цвета на поле, по горизонтали, вертикали или диагонали окажутся фишки противника, эти фишки обращаются в цвет игрока, и ход считается результативным.
2. Каждая выставленная фишка должна совершать результативный ход (обращать хотя бы одну фишку противника).
3. Если результативных ходов нет, ход передается противнику.
4. Игра заканчивается, когда заняты все 64 клетки поля.
5. Победителем считается тот, фишек чьего цвета на поле больше.

На экране должен отображаться текущий счет игры после каждого хода; должно сообщаться, чей сейчас ход, также можно отобразить время игры. По окончании игры показывается, кто победил и финальный счет.

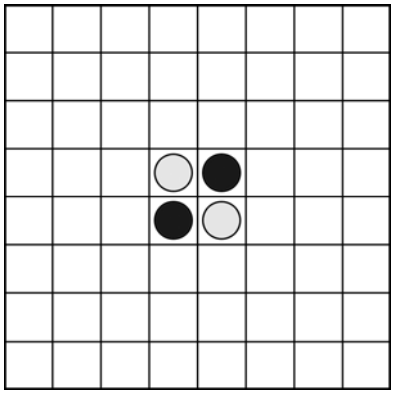
Подсказки по выполнению

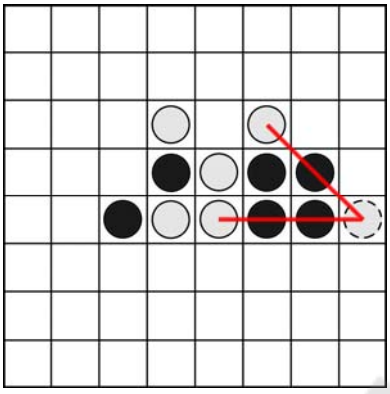
Для проверки того, какие фишки после хода обращать, а какие нет, можно воспользоваться следующим алгоритмом:

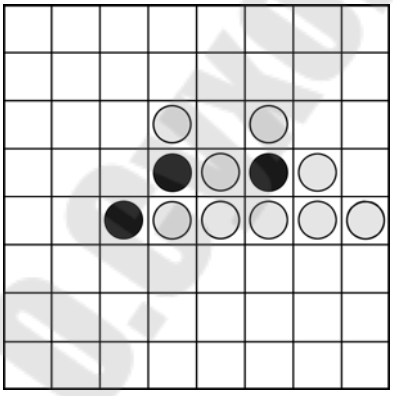
От выставленной фишки есть восемь направлений проверки.

1. Выбрать первое направление.
2. Последовательно проверять клетки данного направления, перемещаясь по клеткам, занятым фишками противника.
3. Завершать проверку направления, если встретилось пустое поле, либо край доски, либо фишка того же цвета, что и выставленная. В последнем случае передавать флаг о том, что фишки в данном направлении надо обратить, и пройти по данному направлению еще раз, обращая фишки противника.

4. Выбрать следующее направление.
5. Выполнять пункты 2-4, пока не будут проверены все направления.


Начальное положение


Белые ставят шашку


Шашки обращены

Вариант 3 Игра «Пять в ряд»

Правила игры. Играют двое, на поле 13х13 клеток. Один играет крестиками, другой ноликами. Игроки по очереди делают ходы, выставляя свой значок на поле. Победит тот, кто сможет построить непрерывную цепь из пяти своих значков в одну линию по горизонтали, вертикали или диагонали.

Если одному из игроков это удалось, то победная цепь выделяется (жирным штифтом, мигает, либо зачеркивается), и на экране отображается имя победителя. Желательно наличие счетчика ходов и времени игры.

Подсказки по выполнению. Крестик и нолик – элементы одного класса!

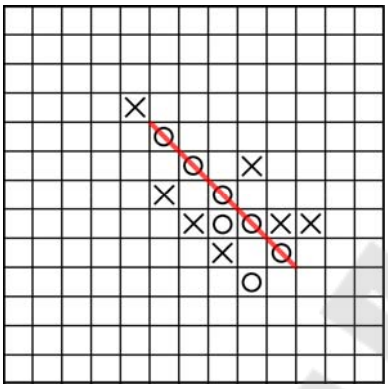
Для проверки того, построена цепь или нет, можно воспользоваться следующим алгоритмом:

От выставленного значка есть четыре пары направлений проверки в две противоположных стороны: вверх и вниз, влево и вправо, по диагонали влево-вверх и вправо-вниз; по диагонали вправо-вверх и влево-вниз.

1. Выбрать первую пару направлений
2. Выбрать, с какой стороны начать.
3. Сделать счетчик своих элементов равным единице.
4. Последовательно проверять клетки в выбранную сторону, с каждой клеткой, занятой своим значком, увеличивая счетчик. Завершить проверку направления, если встретился значок противника, либо край доски, либо пустое поле.
5. Если счетчик достиг значения 5, значит, цепь построена. Завершить программу.

6. Выполнить пункты 3 - 5 в противоположную сторону из данной пары.
7. Выбрать следующую пару направлений
8. Перейти к пункту 2.

Таким образом, если алгоритм не дал победного результата, значит игра продолжается.



«Нолики» победили

Вариант 4 Игра «Поддавки»

Правила игры. Играют два игрока на стандартной шахматной доске. Один играет белыми шашками, другой черными. Расстановка стандартная для русских шашек (первые три линии с каждой стороны, на черных клетках). Ходят по очереди только по черным клеткам. Рубить обязательно. Срубленная шашка убирается с доски. Выигрывает тот игрок, у которого не осталось шашек на доске.

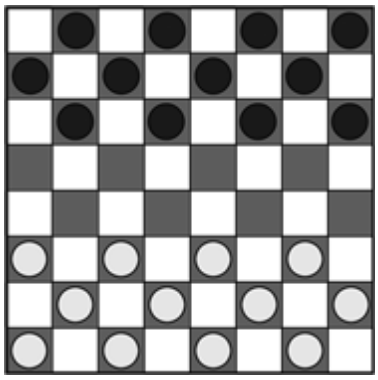
Ход: Игрок перемещает свою шашку на соседнюю клетку по диагонали, не занятую своей шашкой либо шашкой противника. Ходить можно только вперед (от своего края доски). Если шашка достигла противоположного края доски, она снимается с доски (дамок нет).

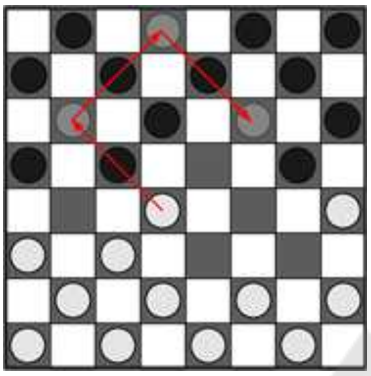
Рубка: Если в момент хода, рядом с шашкой игрока находится шашка противника, и позади нее в том же направлении есть свободная клетка, то шашка игрока «перепрыгивает» через шашку противника, «срубая» ее, и перемещаясь на следующую клетку после нее клетку. Если из нового положения ЭТОЙ ШАШКЕ опять есть что срубить, то рубка повторяется. Рубить можно и вперед и назад. Если в процессе рубки нескольких шашек подряд, шашка окажется на противоположном краю доски, она не убирается, а продолжает рубить в обратном направлении.

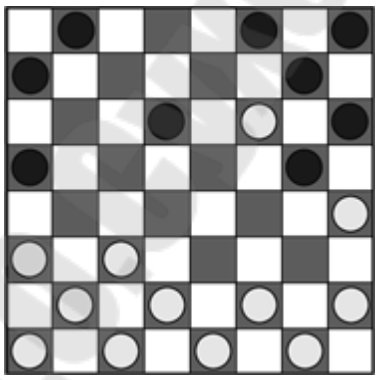
Если в момент хода имеется несколько вариантов рубки, то игрок выбирает любой из них.

Рубка считается сделанным ходом, и по ее окончании ход переходит к противнику.

Программа должна следить за соблюдением правил игры.


Начальное положение


Белые обязаны рубить


Три шашки срублены

Вариант 5 Игра «Паззл»

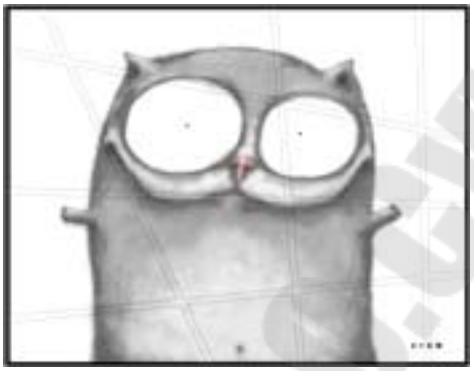
Правила игры. На игровое поле загружается картинка (любая по желанию игрока). Затем она разделяется прямыми непересекающимися линиями на некоторое количество четырехугольных кусочков (желательно сделать несколько уровней сложности с разным числом кусков). Данные кусочки перемешиваются случайным образом, образуя хаотичную кучу в правой части экрана. Задача игрока – перемещая кусочки из правой части экрана в центр, и прикладывая друг к другу, собрать первоначальную картинку. В любой момент можно переместить любой кусочек либо группу объединенных кусочков. После получения первоначальной картинки, игра заканчивается.

Подсказки по выполнению. Для хранения каждого кусочка можно использовать отдельный компонент из визуальной библиотеки. Места разрезов на каждом кусочке можно закодировать определенным кодом, так, чтобы по обе стороны от разреза были одинаковые коды. Тогда проверка на правильность соединения сводится к проверке кодов сторон.

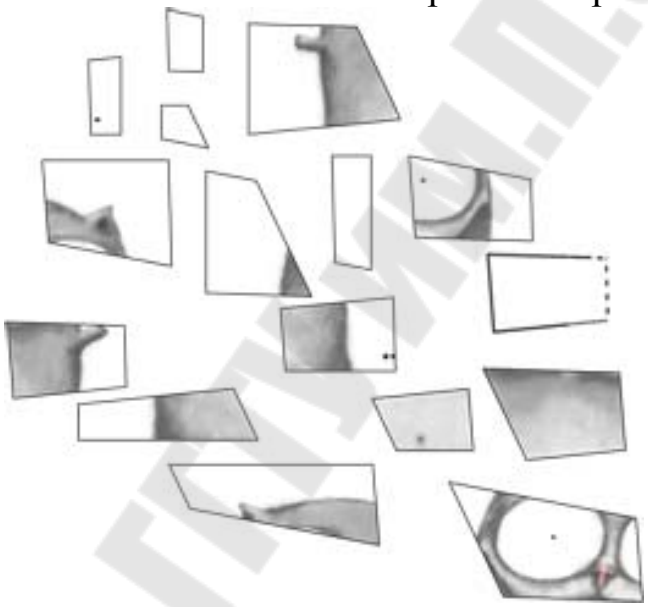
Еще один алгоритм проверки правильности соединения заключается во внесении всех кусочков в массив, и проверке номеров элементов данного массива. Если элементы смежны (один индекс имеет то же значение, а второй отличается на единицу, например (4,5) и (4,6)), то значит соединение верно.

Если два кусочка соединены верно (расстояние между сторонами соединения на экране меньше какой-то величины, они «склеиваются» - вместо двух объектов остается один, содержащий в себе изображения

обоих кусочков, либо просто два объекта «привязываются друг к другу и перемещаются вместе).



Картинка разбивается на части Собранная картинка



Разрозненные кусочки картинки

Вариант 6 Игра «Питон»

Правила игры. По замкнутому прямоугольному полю, состоящему из невидимых клеток, перемещается питон. Питон состоит из головы, размером в одну клетку, и хвоста, первоначально размером в пять клеток. Сразу после появления, голова питона начинает движение прямо (в сторону от хвоста), с постоянной скоростью. Хвост движется строго за головой, перемещаясь по тем клеткам, где была голова. Если игрок нажимает левую кнопку мыши (или клавишу «влево»), питон поворачивает на 90 градусов против часовой стрелки; при нажатии правой кнопки мыши (клавиши «вправо») – питон поворачивает на 90 градусов по часовой стрелке, и продолжает движение в новом направлении.

Каждые несколько секунд длина питона увеличивается на одну клетку.

На экране в случайных местах появляются «кролики» - случайные цифры от 1 до 9. Если питон «наедет» головой на кролика, то считается, что он его съел – счет игры увеличивается на величину значения съеденной цифры. Иногда на экране появляются мангусты – значки с буквой «М». Если питон «съест» мангусту, либо собственный хвост, либо границу экрана он умирает и игра заканчивается.

Кроме игрового поля на экране должен отображаться текущий счет, время игры, и текущая длина питона.

Подсказки по выполнению. Суть движения питона состоит в том, что каждую единицу времени его голова перемещается на клетку вперед; там, где она находилась, рисуется элемент тела, а последний элемент хвоста стирается. Остальное тело не меняется и не перерисовывается. Процесс роста реализуется так же, только последний элемент хвоста стирать не надо – получится, что голова на одну единицу сместилась, а хвост остался на месте, то есть питон вырос на 1 клетку.

Движение против часовой стрелки: вправо, вверх, влево, вниз. По часовой: вправо, вниз, влево, вверх. При нажатии кнопки поворота, направление движения меняется циклично.

Не забудьте проверить, чтобы ваши кролики и мангусты не появлялись на уже занятых клетках!

Вариант 7 Игра «Сапер»

Правила игры. На квадратном поле 16х16 клеток случайным образом разбросаны 40 мин. Вокруг каждой мины находятся цифры – каждая цифра точно показывает, сколько вокруг нее мин в клетках, смежных по горизонтали, вертикали и диагонали. Если вокруг клетки мин нет, она пуста.

В начале игры поле закрыто. Игрок начинает проверять произвольные клетки, открывая их содержимое. Если он думает, что под данной клеткой – мина, он может не открывать ее, а маркировать флажком. Если игрок откроет пустую клетку (вокруг которой ни одной мины), то программа автоматически открывает все пустые клетки, смежные с данной, а также (не все!) не занятые минами клетки, смежные с пустыми.

Задача игрока – открыть все клетки поля, не занятые минами, и маркировать все мины. Если игрок попытается открыть заминированную клетку, он проигрывает.

На экране должно отображаться время игры и количество немаркированных мин.

Программа должна позволять игроку варьировать начальный размер поля и количество мин на нем.

Подсказки по выполнению.

Проще всего заполнить игровое поле, сперва случайным образом раскидав по нему мины, а потом посчитав для каждой клетки количество мин в смежных клетках.

Все поле открыто

Игрок нарвался на мину

Вариант 8 Игра «Аквариум»

Правила игры. Экран представляет собой аквариум, заполненный водой. В аквариуме живут карпы. Каждый карп умеет перемещаться внутри аквариума, разворачиваться возле стенок аквариума, и сканировать пространство перед собой в некотором секторе. Карпов в аквариуме много, и они плавают стаями. Также в аквариуме живет щука. Она движется быстрее любого карпа, но хуже видит (меньше радиус сканирования). Увидев карпа, щука начинает преследование, настигнув – съедает. Съеденный карп удаляется из аквариума (а его объект уничтожается). Карпы, увидев щуку, начинают движение в противоположную сторону (убегают). Программа завершает работу, когда щука съест последнего карпа.

Подсказки по выполнению. Аквариум – это класс-контейнер; содержит в себе класс «рыба», к которому относятся и карп и щука. Чтобы создать стаю, карпов можно объединить в связный список. При съедании карпа, его объект уничтожается и удаляется из списка.

Описать движение рыб можно просто: движется по прямой, потом по случайному событию меняет направление движения в случайную сторону.

Для организации процесса «сканирования» пространства, можно координаты всех рыб хранить в отдельном массиве и брать информацию о местоположении оттуда. Если положение какого-нибудь

карпа находится в радиусе зрения щуки, она начинает преследование – вместо движения по прямой начинает движение кратчайшим путем к текущему положению карпа.

Вариант 9 Игра «Морской бой»

Правила игры. Играет один человек против компьютера. На двух полях 10х10 клеток случайным образом расставляются корабли: 1 четырехпалубный, 2 трехпалубных, 3 двухпалубных, 4 однопалубных. Между любыми двумя кораблями на поле обязана быть хотя бы одна свободная клетка.

Поле, принадлежащее компьютеру, закрыто.

Игрок и компьютер начинают по очереди делать ходы, «стреляя» по клеткам поля противника. Если на «простреленной» клетке находится часть корабля, то на экране мигает надпись «ранен», и эта клетка выделяется оранжевым цветом. Если все палубы корабля пробиты, мигает надпись «убит», клетки корабля закрашиваются красным, а клетки вокруг корабля зачеркиваются крестом (по правилам там не может находиться другой корабль).

Компьютер делает ход случайным образом. Однако если он «ранил» корабль игрока, то он начинает обстреливать область вокруг этой клетки, пока корабль не будет потоплен. На поле игрока раненые и убитые корабли отображаются так же, как и на поле компьютера. Кроме того, ни компьютер, ни игрок не могут два раза стрелять в одну и ту же клетку, и в клетки вокруг потопленных кораблей, зачеркнутые крестом.

Выиграет тот, кто первым потопит все корабли противника.

Подсказки к выполнению. Все корабли – экземпляры одного класса, отличающиеся только количеством палуб.

Примерный алгоритм случайной расстановки кораблей на поле:

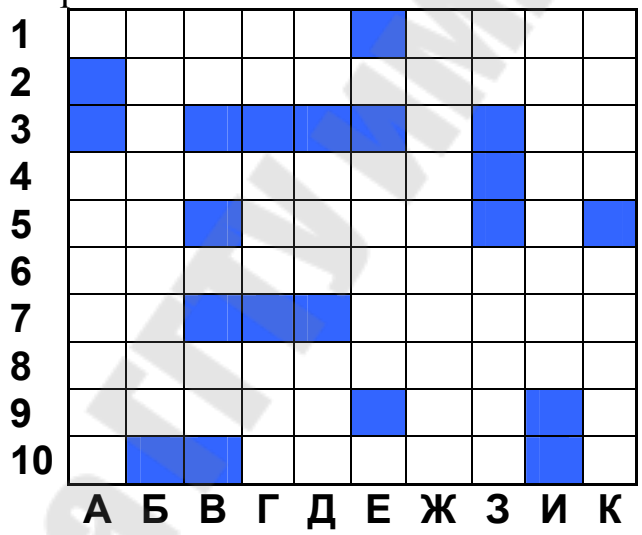
Расставлять корабли надо по размеру, в убывающем порядке, начиная с четырехпалубного. Текущее расположение может храниться в матрице 10х10, где занятые кораблями клетки, а также клетки вокруг (на них тоже нельзя размещать другие корабли), обозначены единицей, свободные клетки – нулем. Для каждого корабля выполняется следующий алгоритм (описан на примере четырехпалубного):

1. Начать проверять клетки поля в горизонтальном направлении с первой горизонтали.
2. Если данная клетка свободна, то проверить следующие за ней три клетки (итого 4 клетки под 4 палубы).
3. Если там тоже свободно, то значит сюда теоретически можно поставить корабль – сохранить координаты данной клетки в

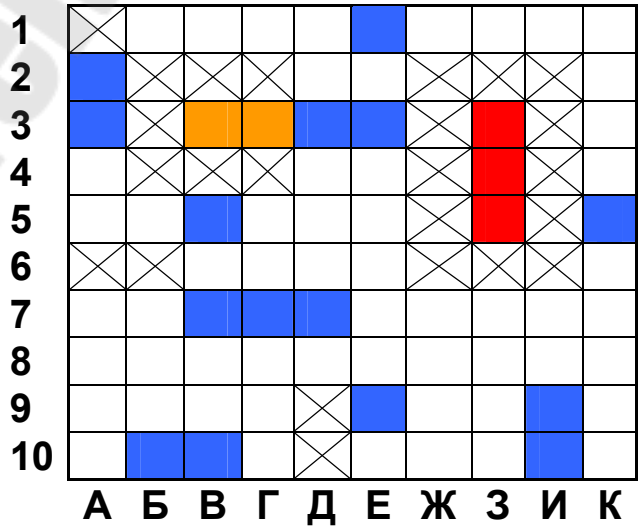
нумерованном списке, с пометкой, что это горизонтальное расположение.

4. Перейти к следующей клетке этой горизонтали. Повторять пункты 2 и 3 до седьмой клетки включительно (дальше проверять бессмысленно, так как корабль просто не войдет).
5. Перейти на следующую горизонталь.
6. Выполнить аналогичный алгоритм для вертикалей.
7. Из получившегося списка разрешенных координат выбрать случайным образом одну координату.
8. Разместить корабль, начиная с выбранной координаты, в соответствующем направлении.
9. Внести изменения в матрицу занятых клеток, добавив туда клетки, занятые кораблем, и клетки вокруг корабля.
10. Обнулить связный список

И так для всех кораблей.



Пример начальной расстановки кораблей



Трехпалубный подбит, четырехпалубный ранен

Вариант 10 Игра «Двойной Теннис»

Правила игры. Играют двое игроков. Экран представляет собой игровое поле, ограниченное слева и справа. Ракетки игроков расположены сверху и снизу экрана, и выглядят по-разному.

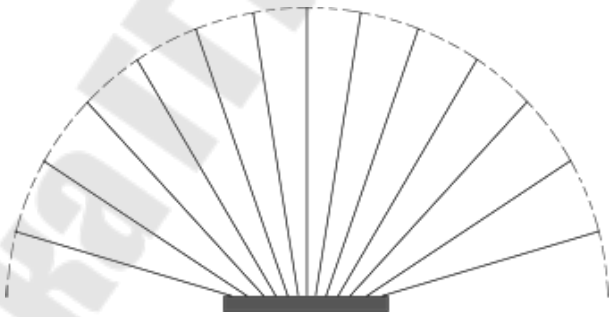
В начале игры ракетки находятся посередине экрана, и на каждой ракетке находится по мячу разного цвета, которые после нажатия клавиши «старт» одновременно вылетают под определенным углом (см. ниже) в сторону противника.

Перемещая свою ракетку в горизонтальной плоскости, каждый игрок должен подставлять свою ракетку под летящие в его сторону мячи, отбивая их. Если мяч пролетел мимо ракетки, игроку-сопернику начисляется призовое очко, а мячи и ракетки возвращаются на исходные позиции. Игра ведется до заранее заданного количества очков.

Скорость перемещения ракеток постоянна, скорость перемещения мячей изначально одинакова и постепенно растет.

От стен и друг от друга мячи отражаются по правилам оптики – под тем же углом, что и упали.

Угол отражения мяча от ракетки зависит не от угла падения, а от места касания ракетки.



Мяч отразится от ракетки в соответствующем направлении

3.2 Вопросы для самоконтроля

Вопросы см в работе №2

4 РАБОТА С ФАЙЛАМИ И КАТАЛОГАМИ

Цель: *получить навыки разработки программ, использующих классы File, FileInfo, DirectoryInfo, FileSystemWatcher, FileStream, BinaryReader и BinaryWriter, научиться осуществлять сериализацию и десериализацию объектов.*

4.1 Краткие теоретические сведения

В пространстве имен *System.IO* определены классы *File* и *FileInfo***,** с помощью которых можно выполнять создание, удаление, перемещение файлов, а также получение их свойств.

Для наблюдения за изменениями файлов и каталогов определен класс *FileSystemWatcher***.**

Поток – последовательность данных. Не зависит от устройства, с которым происходит обмен данными.

Классы-потоки:

- *Stream* **-** абстрактный, базовый для остальных потоков класс;
- *FileStream* **-** поток для чтения и записи файлов на диске;
- *NetworkStream* **-** поток для чтения и записи байтов через сеть из любого процесса;
- *MemoryStream* **-** поток для чтения и записи данных из области памяти;
- *BufferedStream* **-** поток для чтения и записи данных из потока;
- *CryptoStream* **–** потоковая реализация криптографии**.**

Классы, производные от *Stream*, работают с нетипизированными данными. Это удобно для перемещения данных, но не удобно для обработки самих данных.

Для обеспечения структурированного доступа к данным в пространстве *System.IO* определены классы: *BinaryReader* и *BinaryWriter, StreamReader* и *StreamWriter, StringReader* и *StringWrite**r***.

Сериализация – это процесс сохранения объектов.

Десериализация – это восстановление сохраненных объектов.

Объекты можно сохранять в одном из двух форматов:

- двоичном;
- и в виде XML-файлов (SOAP – Simple Object Access Protocol: простой протокол доступа к объектам).

Порядок выполнения сериализации в двоичном формате.

1. Подключить пространство имен System.Runtime.Serialization.Formatters.Binary.
2. Установить для класса, объекты которого нужно сохранять, а также для всех связанных с ним классов атрибут

[Serializable**]**

Те поля, которые не нужно сохранять, нужно пометить атрибутом **[**NonSerializable**]**

Например,

```
[Serializable]   class Man
   { string FIO;
      int GodRogdenija;
 [NonSerialized]
     double zarplata;
public Man(string fam, int GodR, double poluchka)
{FIO=fam; GodRogdenija = GodR; zarplata = poluchka;}   }
```

3. Создать поток и связать его с файлом на диске, в котором будут сохраняться объекты.

4. Создать объект форматирования класса BinaryFormatter, управляющий сериализацией.

5. Вызвать метод Serialize объекта форматирования, чтобы сохранить объекты в потоке.

6. Закрыть поток.

При создании объекта форматирования можно использовать конструктор без параметров:

```
BinaryFormatter bf = new   BinaryFormatter();
```

Метод *Serialize(Stream strm, Object ob) (mun void***) -** сериализует объект или граф объектов с заданной вершиной (корнем) в заданном потоке.

Например,

```
  Man  chel = new    Man("Иванов", 1979, 200000);
FileStream   fs  =  new  FileStream("file.dat",
FileMode.Create);
BinaryFormatter   bf =    new   BinaryFormatter();
    bf. Serialize(fs, chel);
    fs.Close();
```

Порядок выполнения десериализации.

1. Открыть поток для доступа к сериализованного объекта.

2. Создать объект форматирования класса *BinaryFormatter*, управляющий сериализацией.

3. Вызвать метод *Deserialize* объекта форматирования и привести его результат к типу десериализуемого объекта.

Метод *Deserialize(Stream strm) (mun результата Object)* восстанавливает из потока *strm* объект или граф объектов. Возвращает значение объекта или корневого объекта графа.

Например,

```
     FileStream  fs  =  new  FileStream("file.dat",
FileMode.Open);
 BinaryFormatter   bf = new   BinaryFormatter();
Man   chel = (Man) bf.Deserialize(fs); fs.Close();
```

Для успешной десериализации необходимо, чтобы текущая позиция в потоке находилась в начале графа объекта.

Класс *BinaryFormatter* лучше применять для сохранения объектов, используемых внутри одного приложения, так как он использует эффективное двоичное представление информации.

Если объекты планируется использовать в других приложениях, лучше воспользоваться форматом SOAP.

Порядок выполнения сериализации в формате SOAP.

1. Подключить пространство имен
`System.Runtime.Serialization.Formatters.Soap.`
2. Установить для класса, объекты которого нужно сохранять, а также для всех связанных с ним классов атрибут
`[Serializable]`
3. Создать поток и связать его с файлом на диске, в котором будут сохраняться объекты.
4. Создать объект форматирования класса SoapFormatter, управляющий сериализацией.
5. Вызвать метод Serialize объекта форматирования, чтобы сохранить объекты в потоке.
6. Закрыть поток.

При создании приложений, в которых используется пространство имен System.Runtime.Serialization.Formatters.Soap, необходимо ссылаться на сборку
`System.Runtime.Serialization.Formatters.Soap.dll.`
Для этого нужно выполнить команду меню
`Project – Add Reference`

Пример 4.1 Требуется сохранять информацию о студентах, размещенную в таблице, в файле на диске, используя сериализацию.

На рисунке 4.1 представлено окно формы, а код программы – на рисунке 4.2

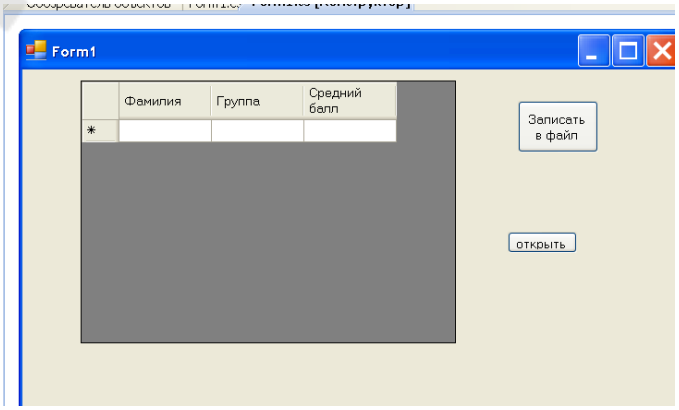


Рисунок 4.1 – Окно формы

```
[Serializable] struct Student
    {
string fam, gruppa;
        double sb;
public void readFromTable(DataGridViewRow dr)
        {
            fam = dr.Cells[0].Value.ToString();
            gruppa = dr.Cells[1].Value.ToString();
            sb = Convert.ToDouble(dr.Cells[2].Value);
        }
        public void writeToTable(DataGridViewRow dr)
        {dr.Cells[0].Value = fam;
            dr.Cells[1].Value = gruppa;
            dr.Cells[2].Value = sb;
        }

    }
private void button1_Click(object sender, EventArgs e)
        {
Student st = new Student();
if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
FileStream stream = File.Create(saveFileDialog1.FileName);
BinaryFormatter bf = new BinaryFormatter();
int n = dataGridView1.RowCount - 1;
for (int i = 0; i < n; i++)
                {
st.readFromTable(dataGridView1.Rows[i]);
bf.Serialize(stream, st);
                }
                stream.Close();
}}}
```

Рисунок 4.2 – Листинг программы примера 4.1

Пример 4.2 Требуется получить информацию о студентах из файла на диске, используя десериализацию, и вывести ее в таблицу.

Код программы представлен на рисунке 4.3.

```
Student   st = new   Student();
if   (openFileDialog1.ShowDialog() == DialogResult.OK)
       {
FileStream   open_stream =
File.Open(openFileDialog1.FileName,FileMode.Open);
BinaryFormatter bf = new BinaryFormatter();
int   i=0;
 bool   f = true;

while (f)
    {
try
   {
st = (Student)  bf.Deserialize(open_stream);
dataGridView1.Rows.Add( );
st.writeToTable(dataGridView1.Rows[i]);
i++;
}
catch (SerializationException) { f = false; }
}
open_stream.Close()
```

Рисунок 4.2 – Листинг программы примера 4.2

4.2 Варианты заданий

Вариант 1

Разработать приложение, выполняющее следующие функции:

- Предоставление пользователю возможности выбора текущей папки из списка заранее заданных.
- Вывод дерева папок и файлов для выбранной папки с использованием элемента управления *treeView*.
- Возможность создания новой папки внутри текущей по нажатию соответствующей кнопки.
- Открытие и вывод любого текстового файла из текущей папки с использованием элементов класса *File* **(**открывать файлы с использованием *OpenFileDialog***).**

Вариант 2

Разработать приложение, выполняющее следующие функции:

- Вывод списков файлов двух выбранных папок в виде таблицы (например, используя элемент управления DataGridView)

- Предоставление возможности удалять и копировать в соседнюю (по таблице) выбранные файлы по нажатию соответствующих кнопок.
- Получение и вывод информации о выбранном файле (время создания, размер, атрибуты, с использованием элементов класса *FileInfo*.

Вариант 3

Разработать приложение, выполняющее следующие функции:

- Вывод дерева папок и файлов для заданной папки с использованием элемента управления *treeView*.
- Возможность создания новой папки внутри текущей и удаления выбранной в дереве папки.
- Открытие и вывод выбранного в дереве текстового файла из текущей папки с использованием элементов класса *File***.**
- Отслеживание переименования файлов в заданной папке (в случае переименования выдавать сообщение) с помощью класса *FileSystemWatcher*.

Вариант 4

Разработать приложение, выполняющее следующие функции:

- Запись внесенной в таблицу информации о фильмах в структурированный файл с помощью класса *BinaryWriter*.

Структура записи в файл:

- *название*
- *фамилия режиссера*
- *жанр*
- *количество занятых в съемке актеров.*

- Считывание информации из файла с помощью класса *BinaryReader* и вывод в виде таблицы.
- Изменение любой записи в таблице и сохранение измененной строки таблицы в соответствующем месте файла, осуществляя прямой доступ**.**
- При выполнении любого из описанных выше действий выдача информации о размере файла и времени последнего сохранения.

Вариант 5

Разработать приложение, выполняющее следующие функции:

- Запись внесенной в таблицу информации об авиарейсах в файл с применением сериализации в *двоичном* формате.

Структура записи в файл:

- *номер рейса*
- *пункт отправления*
- *пункт назначения*
- *цена билета.*

- Считывание информации из файла с применением десериализации и вывод в виде таблицы.
- Поиск рейса с минимальной ценой на билет, увеличение этой цены на 25% и сохранение изменений в файле.
- Предоставление возможности смены рабочего каталога.

Вариант 6

Разработать приложение, выполняющее следующие функции:

- Запись внесенной в таблицу информации о спортсменах в файл с применением сериализации в формате *Soap*.

Структура записи в файл:

- *Фамилия*
- *Год рождения*
- *вид спорта*
- *рейтинг.*

- Считывание информации из файла с применением десериализации и вывод в виде таблицы.
- Сохранение информации о спортсменах, рейтинг которых превышает заданный в *текстовом* файле.

Вариант 7

Разработать приложение, выполняющее следующие функции:

- Предоставление пользователю возможности выбора текущей папки из списка заранее заданных.
- Вывод дерева папок и файлов для выбранной папки с использованием элемента управления *treeView.*
- Возможность удаления файлов и пустых папок из текущей папки по нажатию соответствующей кнопки.
- Открытие и вывод любого текстового файла из текущей папки с использованием элементов класса *File* **(**открывать файлы с использованием *OpenFileDialog*).

Вариант 8

Разработать приложение, выполняющее следующие функции:

- Вывод списков файлов двух выбранных папок в виде таблицы (например, используя элемент управления *DataGridView*)
- Предоставление возможности сравнивать выбранные в соседних столбцах таблицы файлы по нажатию соответствующей кнопки с выводом результата сравнения в процентах (использовать *FileStream***)**.
- Получение и вывод информации о выбранном файле (время создания, размер, с использованием элементов класса *FileInfo*.

Вариант 9

Разработать приложение, выполняющее следующие функции:

- Вывод дерева папок и файлов для заданной папки с использованием элемента управления *treeView*.
- Возможность создания новой папки внутри текущей и удаления выбранной в дереве папки.
- Открытие и вывод выбранного в дереве текстового файла из текущей папки с использованием элементов класса *File***.**
- Отслеживание переименования файлов в текущей папке (в случае переименования выдавать сообщение) с помощью класса *FileSystemWatcher*.

Вариант 10

Разработать приложение, выполняющее следующие функции:

- Запись внесенной в таблицу информации о фильмах в структурированный файл с помощью класса *BinaryWriter*.

Структура записи в файл:

- *название*
- *фамилия режиссера*
- *количество серий*
- *жанр.*

- Считывание информации из файла с помощью класса *BinaryReader* и вывод в виде таблицы.
- Изменение любой записи в таблице и сохранение измененной строки таблицы в соответствующем месте файла, осуществляя прямой доступ**.**

- При выполнении любого из описанных выше действий выдача информации о размере файла и времени последнего сохранения.

Вариант 11

Разработать приложение, выполняющее следующие функции:

- Запись внесенной в таблицу информации о поездах в файл с применением сериализации в двоичном формате.

Структура записи в файл:

- *номер поезда*
- *пункт отправления*
- *пункт назначения*
- *цена плацкарты*
- *количество свободных мест.*

- Считывание информации из файла с применением десериализации и вывод в виде таблицы.
- Поиск поезда с максимальным количеством свободных мест, увеличение этой цены плацкарты на 20% и сохранение изменений в файле.
- Предоставление возможности смены рабочего каталога.

Вариант 12

Разработать приложение, выполняющее следующие функции:

- Запись внесенной в таблицу информации о школьниках в файл с применением сериализации в формате *Soap.*

Структура записи в файл:

- *Фамилия*
- *Год рождения*
- *номер школы*
- *Оценки за экзамены по математике и физике* .

- Считывание информации из файла с применением десериализации и вывод в виде таблицы.
- Сохранение информации о школьниках, сдавших экзамены на 9 и 10, в *текстовом* файле.

Вариант 13

Разработать приложение, выполняющее следующие функции:

- Запись внесенной в таблицу информации об авиарейсах в файл с применением сериализации в двоичном формате.

Структура записи в файл:

- *номер рейса*
- *пункт отправления*
- *пункт назначения*
- *цена билета.*

- Считывание информации из файла с применением десериализации и вывод в виде таблицы.
- Поиск рейса с минимальной ценой на билет, увеличение этой цены на 25% и сохранение изменений в файле.
- Предоставление возможности смены рабочего каталога.

Вариант 14

Разработать приложение, выполняющее следующие функции:

- Запись внесенной в таблицу информации о фильмах в структурированный файл с помощью класса *BinaryWriter*.

Структура записи в файл:

- *название*
- *фамилия режиссера*
- *жанр*
- *количество занятых в съемке актеров.*

- Считывание информации из файла с помощью класса *BinaryReader* и вывод в виде таблицы.
- Изменение любой записи в таблице и сохранение измененной строки таблицы в соответствующем месте файла, осуществляя прямой доступ**.**
- При выполнении любого из описанных выше действий выдача информации о размере файла и времени последнего сохранения.

Вариант 15

Разработать приложение, выполняющее следующие функции:

- Запись внесенной в таблицу информации о спортсменах в файл с применением сериализации в формате **Soap**.

Структура записи в файл:

- *Фамилия*
- *Год рождения*
- *вид спорта*
- *рейтинг.*

- Считывание информации из файла с применением десериализации и вывод в виде таблицы.

- Сохранение информации о спортсменах, рейтинг которых превышает заданный в *текстовом* файле.

4.3 Вопросы самоконтроля

1. Просмотр и выбор файлов с помощью OpenFileDialog. Свойства.
2. Методы класса OpenFileDialog.
3. Класс File.
4. Класс FileInfo.
5. Работа с каталогами. Класс Directory.
6. Работа с каталогами. Класс DirectoryInfo.
7. Отслеживание изменений в файлах и каталогах.
8. Потоки ввода-вывода.
9. Класс Stream.
10. Класс FileStream.
11. Классы BinaryReader и BinaryWriter.
12. Порядок выполнения сериализации в двоичном формате.
13. Порядок выполнения десериализации.
14. Порядок выполнения сериализации в формате SOAP

5 КОЛЛЕКЦИИ СТРУКТУР ДАННЫХ. КЛАССЫ -ПРОТОТИПЫ

Цель работы: *Изучить возможность использования коллекций пространства имен System.Collection. Научиться создавать и использовать классы-прототипы для хранения и обработки данных.*

5.1 Краткие теоретические сведения

Класс *ArrayList* предназначен для поддержки динамических массивов, которыепри необходимости могут увеличиваться или сокращаться.

Объект класса *ArrayList* представляет собой массив переменной длины,элементами которого являются объектные ссылки. Любой объект класса *ArrayList* создается с некоторым начальным размером. При превышении этого размера коллекция автоматически его увеличивает. В случае удаления объектов массив можно сократить.

Класс *ArrayList* реализует интерфейсы *ICollection*, *IList*, *IEnumerable* и *ICloneable*.

В классе ArrayList определены следующие конструкторы:

```
public ArrayList()
public ArrayList(ICollection c)
public ArrayList(int capacity)
```

Класс *Hashtable* предназначен для создания коллекции, в которой для хранения объектов используется *хеш-таблица.* В хеш-таблице для хранения информации используется механизм, именуемый *хешированием* (hashing).

В классе *Hashtable* определено множество конструкторов, включая следующие (они используются чаще всего):

```
public Hashtable()
public Hashtable(IDictionary c)
public Hashtable(int capacity)
public Hashtable(int capacity, float fillRatio)
```

Пример 5.1

```
public virtual ICollection Keys { get; }
public virtual ICollection Values { get; }

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;
namespace ConsoleApplication1
{class Program
    {static void Main(string[] args)
```

```
        {// Создаем хеш-таблицу.
            Hashtable ht = new Hashtable();
            // Добавляем элементы в хеш-таблицу.
            ht.Add("здание", "жилое помещение");
    ht.Add("автомобиль", "транспортное средство");
            ht.Add("книга", "набор печатных слов");
            ht.Add("яблоко", "съедобный фрукт");
//Добавлять элементы можно также С помощью //индексатора,
ht["трактор"] = "сельскохозяйственная машина";
            // Получаем коллекцию ключей.
            ICollection c = ht.Keys;
// Используем ключи для получения значений,
            foreach (string str in c)
   Console.WriteLine(str + ": " + ht[str]);
            Console.ReadLine();
        }}}
```

Класс *Stack* реализует стековую структуру данных. Создание объекта класса *Stack* осуществляется следующим образом:

```
 Stack st = new Stack( );
// создание стека с емкостью по умолчанию (10 элементов)
Stack  st = new  Stack(100);
// создание массива с емкостью 100 элементов
```

Свойство:

- *Count* – количество элементов в стеке.

Методы:

- *Push(object ob)* – добавляет объект ob в верхнюю часть класса (вершину стека).
- *Pop( )* – удаляет «верхний» объект из стека.
- *Peek( )* – возвращает объект (тип object) из вершины стека.
- *Contains(object ob)* – возвращает true, если ob находится в стеке, и false в противном случае.

Существуют «стандартные» классы прототипы, которые называются *параметризованными коллекциями*. Такие коллекции – это уже готовые шаблоны для различных структур данных – стеков, очередей, списков, бинарных деревьев и т.д. Эти коллекции хранятся в пространстве имён *System.Collections.Generic*.

Во всех параметризованных коллекциях имеется так называемый параметр в качестве которого обычно выступает тип данных, с которым работает эта коллекция.

Класс-прототип может содержать произвольное число параметров типа. Для каждого параметра можно задать ограничения, указывающие каким требованиям должен удовлетворять аргумент, соответствующий этому параметру. Например, можно указать, что это будет тип, использующий некоторый интерфейс.

Ограничения задаются после ключевого слова *where*, например:

public class Stack<T>
where T:struct {…}

Здесь задано ограничение, что класс стек может использовать элементы только значимого типа (типа структуры). Для ссылочного типа необходимо использовать ключевое слово *class*.

Указание в качетстве ограничений имени класса означает, что соответствующий параметр (аргумент) должен быть инициализирован или именем этого класса, либо его потомка.

Пример 5.2

Подготовить текстовый файл, каждая строка которого содержит информацию о спортсменах: фамилию, год рождения, вид спорта, разряд.

Описать структуру, содержащую следующие элементы: поля для хранения фамилии, года рождения, вида спорта, разряда, свойство для определения возраста.

Создать класс-прототип для хранения и обработки экземпляров структур, наложить ограничение на параметр типа данных: элементы коллекции должны быть значимого типа, тип –аргумент должен реализовывать интерфейс IComparable.

Написать Windows-приложение, выполняющее следующее функции:

- Считывание данных из файла.
- Вывод информации в виде таблицы.
- Сортировку по убыванию года рождения или по виду спорта (по выбору пользователя)
- Формирование и вывод списка спортсменов старше 18 лет, имеющих II разряд (использовать стандартную коллекцию *ArrayList*).

На рис.5.1 показана реализация задачи.

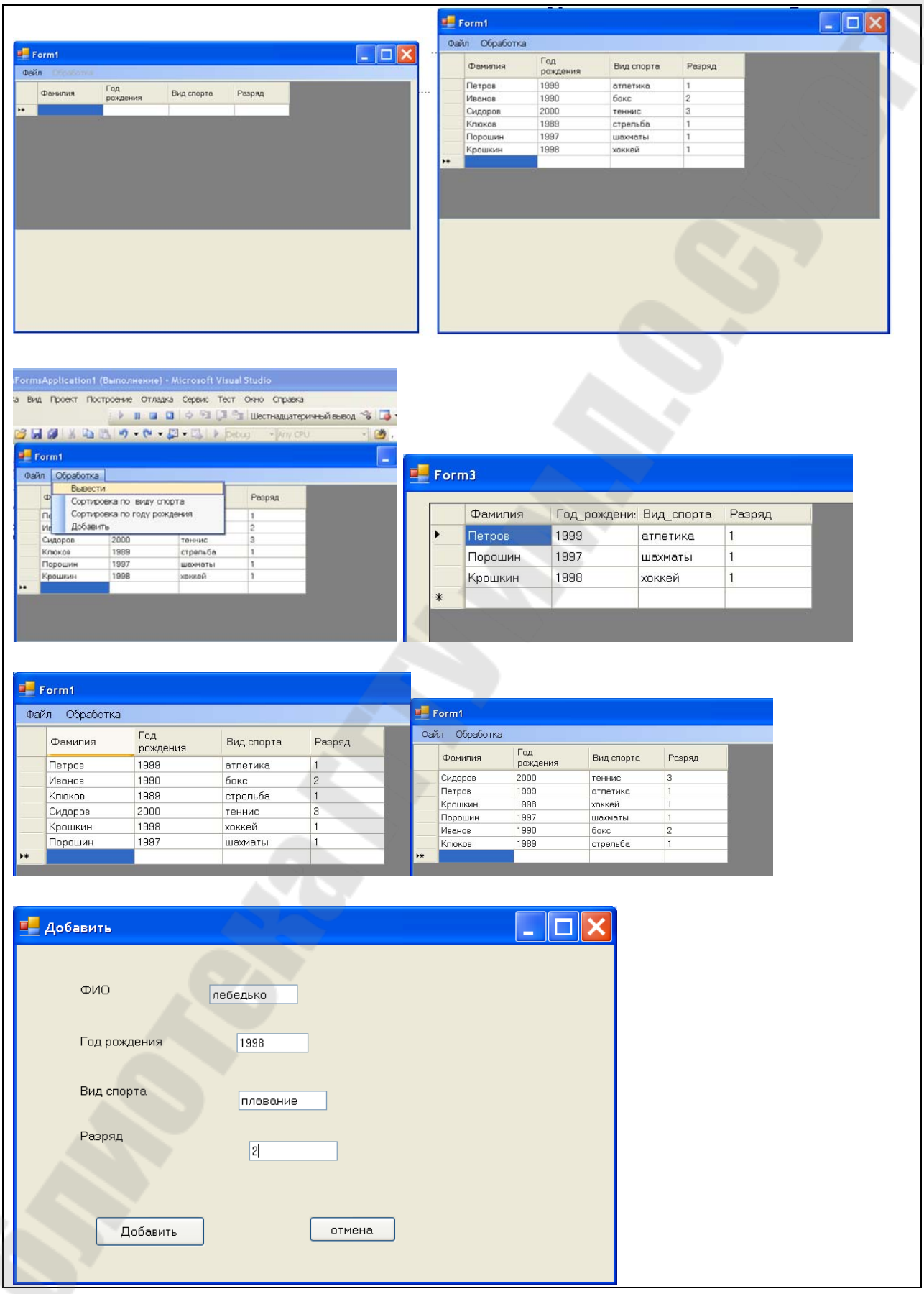


Рисунок 5.1 – Реализация задачи

На рисунке 5.2 представлен код программы.

```
    public struct Sportsman : IComparable
    {
        string fam;
        int god;
        string sport;
        int razr;
        public static int sorting_type = 0;//тип сортировки: 0-
по убыванию года,1- по категории
        public Sportsman(string f, string g, string s, string r)
        {
            fam = f;
            god = Convert.ToInt32(g);
            sport = s;
            razr = Convert.ToInt32(r);
        }
public int Vozrast
        {
            get
            { return DateTime.Now.Year - god; }
        }
        public string Fam
        {get
            { return fam; }
            set { fam = value; }
        }
        public int God
        {
            get
            { return god; }
            set { god = value; }
        }
public string Sport
        {get
            { return sport; }
            set { sport = value; }}
        public int Razr
        {get
            { return razr; }
            set { razr = value; }}
public int CompareTo(object ob)//сортировка
        {Sportsman s = (Sportsman)ob;
            if (sorting_type == 0)
            {
                if (god <= s.god) return 1;
                else return -1;
            }
            else
            {
          if (sport.CompareTo(s.sport) >= 0) return 1;
                else
                    return -1;}}
```

Рисунок 5.2 – Код программы

```
public class Lt<T> where T : struct, IComparable
    {
        T[] items = new T[0];
        public T[] getT
        { get { return items; } }
        public void AddElement(T elem)
        {
      Array.Resize(ref items, items.Length + 1);
            items[items.Length - 1] = elem;
        }
public void RemElement(T elem)
        {
            Array.Resize(ref items, items.Length - 1);
        }
        public int Length
        { get { return items.Length; } }
        public T this[int i]
        {
            get { return items[i]; }
            set { items[i] = value; }
        }
    }
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Collections;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        public Lt<Sportsman> item = new Lt<Sportsman>();
        ArrayList Lt;
private void выходToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Close();
        }
```

Продолжение рисунка 5.2

```
private void добавитьToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Form2 f2 = new Form2();
            f2.f1 = this;
            f2.Show();
        }
private void открытьToolStripMenuItem_Click(object sender,
EventArgs e)
        {
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{StreamReader sr = new StreamReader(openFileDialog1.FileName);
                string[] s;
                Sportsman S;
                int i = 0;
                while (!sr.EndOfStream)
                {
                    dataGridView1.RowCount++;
                    s = (sr.ReadLine()).Split(';');
S = new Sportsman(s[0], s[1], s[2], s[3]);
                    item.AddElement(S);
                    dataGridView1[0, i].Value = s[0];
                    dataGridView1[1, i].Value = s[1];
                    dataGridView1[2, i].Value = s[2];
                    dataGridView1[3, i].Value = s[3];
                    i++;}
                sr.Close();
обработкаToolStripMenuItem.Enabled = true;}}
private void вывестиToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Form3 f3 = new Form3();
            bool f = false;
            int j = 0;
            Lt = new ArrayList();
            for (int i = 0; i < item.Length; i++)
            {
if (item[i].Vozrast < 18 && item[i].Razr == 1)
{
                    f = true;
                    Lt.Add(item[i]);
                    f3.dataGridView1.RowCount++;
f3.dataGridView1[0, j].Value = item.getT[i].Fam;
f3.dataGridView1[1, j].Value = item.getT[i].God;
f3.dataGridView1[2, j].Value = item.getT[i].Sport;
f3.dataGridView1[3, j].Value = item.getT[i].Razr;
                    j++;
                }
            }
```

Продолжение рисунка 5.2

```
            if (f)
                f3.Show();
            else
                MessageBox.Show("Нет таких спортсменов!");
        }
private void
сортировкаПоГодуРожденияToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Sportsman.sorting_type = 0;
            Array.Sort(item.getT);
            int j = 0;
            for (int i = 0; i < item.Length; i++)
            {
                dataGridView1[0, j].Value = item[i].Fam;
                dataGridView1[1, j].Value = item[i].God;
               dataGridView1[2, j].Value = item[i].Sport;
               dataGridView1[3, j].Value = item[i].Razr;
               j++;
            }
        }
private void
сортировкаПоВидуСпортаToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Sportsman.sorting_type = 1;
            Array.Sort(item.getT);
            int j = 0;
            for(int i=0;i<item.Length;i++)
            {
           dataGridView1[0, j].Value = item [i].Fam;
             dataGridView1[1, j].Value = item [i].God;
             dataGridView1[2, j].Value = item [i].Sport;
             dataGridView1[3, j].Value = item[i].Razr;
             j++;
            }

        }
```

Продолжение рисунка 5.2

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form2 : Form
    {
        public Form1 f1;
        public Form2()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
f1.item.AddElement(new Sportsman(textBox1.Text, textBox2.Text,
textBox3.Text, textBox4.Text));
            f1.dataGridView1.RowCount++;
f1.dataGridView1[0, f1.dataGridView1.RowCount - 2].Value =
textBox1.Text;
f1.dataGridView1[1, f1.dataGridView1.RowCount - 2].Value =
textBox2.Text;
f1.dataGridView1[2, f1.dataGridView1.RowCount - 2].Value =
textBox3.Text;
f1.dataGridView1[3, f1.dataGridView1.RowCount - 2].Value =
textBox4.Text;
            this.Close();
        }}}
```

Продолжение рисунка 5.2

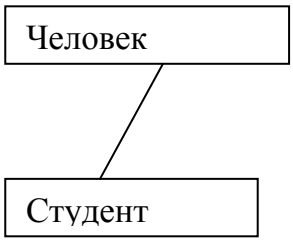
5.2 Варианты заданий

Вариант 1

Подготовить текстовый файл, каждая строка которого содержит фамилию, год рождения, статус, для студентов оценки за сессию, например:

Иванов 1988 студент 5 6 4 3
Петров 1978 министр
Сидоров 1967 преподаватель

Создать иерархию классов:



Класс «человек» должен содержать следующие элементы: поле – фамилия, поле-год рождения, поле статус (студент, бизнесмен, преподаватель и т.д.), метод для получения возраста.

Класс студентов должен содержать дополнительное поле-массив с результатами сдачи сессии, метод для получения среднего балла за сессию.

Создать класс-прототип для хранения и обработки экземпляров классов, наложить ограничение на параметр типа данных: элементы коллекции должны быть типа «человек» или производных от него типов, тип –аргумент должен реализовывать интерфейс *IComparable*.

Написать Windows-приложение, выполняющее следующее функции:

- Считывание данных из файла.
- Вывод информации в виде таблицы.
- Сортировку по возрастанию года рождения
- Формирование и вывод списка студентов, имеющих средний балл ниже заданного (использовать стандартную коллекцию *Stack*).

Вариант 2

Подготовить текстовый файл, каждая строка которого содержит информацию о спортсменах: фамилию, год рождения, вид спорта, разряд.

Описать класс Спортсмен, содержащий следующие элементы: поля для хранения фамилии, года рождения, вида спорта, разряда, свойство для определения возраста.

Создать класс-прототип для хранения и обработки экземпляров структур, наложить ограничение на параметр типа данных: элементы коллекции должны быть значимого типа, тип –аргумент должен реализовывать интерфейс *IComparable.*

Написать Windows-приложение, выполняющее следующее функции:

- Считывание данных из файла.
- Вывод информации в виде таблицы.

- Добавление данных о спортсмене.
- Сортировку по возрастанию года рождения или по виду спорта (по выбору пользователя)
- Формирование и вывод списка спортсменов старше 20 лет, имеющих II разряд (использовать стандартную коллекцию *Stack*).

Вариант 3

Подготовить текстовый файл, содержащий информацию о хранящемся на складе товаре: код товара (целое число), поставщик, цена, количество товара, поступившего на склад (разделителем в файле служит точка с запятой).

Например,

134; ООО «Прогресс»; 25000; 385
134; ИП Петров В.В.; 23000; 200
012; ИП Петров В.В.; 120000; 15

Описать структуру. Структура должна содержать следующие элементы: поля для хранения кода товара, поставщика, цены, количества товара, свойство для определения стоимости товара, свойство для определения наименования товара по коду. Для определения наименования товара по коду необходимо создать перечисление с наименованиями товара, причем для каждого товара определить значение, равное коду товара.

Например,

```
public enum tovar { стул=134, диван=12, шкаф=108 }.
```

Информация в файле должна соответствовать перечислению.

Создать класс-прототип для хранения и обработки экземпляров структур, наложить ограничение на параметр типа данных: элементы коллекции должны быть значимого типа, тип – аргумент должен реализовывать интерфейс *IComparable*.

Написать Windows-приложение, выполняющее следующее функции:

- Считывание данных из файла.
- Вывод информации в виде таблицы.
- Добавление данных о товаре.
- Сортировку по убыванию года цены или по названию (по выбору пользователя)
- Формирование и вывод списка товаров с ценой не выше заданной (использовать стандартную коллекцию *ArrayList*).

Вариант 4

Подготовить текстовый файл, содержащий информацию о запланированных мероприятиях: мероприятие, дата проведения, место проведения (разделителем в файле служит точка с запятой).
Описать структуру. Структура должна содержать следующие элементы: поля для хранения мероприятия, даты проведения и места проведения, свойство для определения количества оставшихся до мероприятия дней. Для описания первого поля необходимо создать перечисление (например, пусть meropr это перечисление с константами Встреча, Концерт, Круглый_стол, тогда поле с мероприятием должно быть описано так: meropr sobytie;). Информация в файле должна соответствовать перечислению.

Создать класс-прототип для хранения и обработки экземпляров структур, наложить ограничение на параметр типа данных: элементы коллекции должны быть значимого типа, тип –аргумент должен реализовывать интерфейс *IComparable*.

Написать Windows-приложение, выполняющее следующее функции:

- Считывание данных из файла.
- Вывод информации в виде таблицы с указанием количества оставшихся до мероприятия дней.
- Добавление данных о новом мероприятии.
- Сортировку по возрастанию даты проведения
- Поиск мероприятия заданного вида, проводимого в заданный день (использовать стандартную коллекцию *HashTable*)

Вариант 5

Подготовить текстовый файл, содержащий информацию о друзьях: фамилия и инициалы, дата рождения, знак Зодиака, номер телефона (разделителем в файле служит точка с запятой).
Описать структуру. Структура должна содержать следующие элементы: поля для хранения фамилии, даты рождения, знака Зодиака и номера телефона, свойство для определения возраста. Для описания второго поля необходимо создать перечисление (например, пусть Zodiak это перечисление с константами Овен, Рыбы, Дева и т.д., тогда поле со знаком Зодиака должно быть описано так: Zodiak znak;).

Создать класс-прототип для хранения и обработки экземпляров структур, наложить ограничение на параметр типа данных: элементы

коллекции должны быть значимого типа, тип –аргумент должен реализовывать интерфейс *IComparable*.

Написать Windows-приложение, выполняющее следующее функции:

- Считывание данных из файла.
- Вывод информации в виде таблицы с указанием возраста друзей
- Добавление данных о новом друге.
- Сортировку по возрастанию даты рождения
- Вывод списка друзей с указанным знаком Зодиака (использовать стандартную коллекцию *Stack*)

Вариант 6

Подготовить текстовый файл, содержащий информацию о рейсах компании «Аэрофлот»: номер рейса, тип самолета, пункт назначения, дата и время вылета (разделителем в файле служит точка с запятой).
Описать структуру. Структура должна содержать следующие элементы: поля для хранения номера рейса, типа самолета, пункта назначения, даты и времени вылета, свойство для определения количества оставшихся до вылета дней. Для описания второго поля необходимо создать перечисление (например, `пусть Tip это перечисление с константами Боинг-767, Ту-154, Ил-85 и т.д.,` тогда поле с типом самолета должно быть описано так: `Tip vid_samoleta;`). Информация в файле должна соответствовать перечислению.

Создать класс-прототип для хранения и обработки экземпляров структур, наложить ограничение на параметр типа данных: элементы коллекции должны быть значимого типа, тип –аргумент должен реализовывать интерфейс IComparable.

Написать Windows-приложение, выполняющее следующее функции:

- Считывание данных из файла.
- Вывод информации в виде таблицы с указанием количества оставшихся до вылета дней.
- Добавление данных о новом рейсе
- Сортировку по убыванию даты вылета или пункта назначения (по выбору пользователя)
- Поиск рейса заданного типа самолета, вылетающего в заданный день (использовать стандартную коллекцию *HashTable*)

Вариант 7

Подготовить текстовый файл, содержащий информацию о пациентах терапевтического отделения больницы: фамилию, год рождения, дату поступления, диагноз.

Описать структуру. Структура должна содержать следующие элементы: поля для хранения фамилии, даты рождения, даты поступления, диагноза, свойство для определения продолжительности пребывания в больнице в днях, метод для определения возраста. Для описания последнего поля необходимо создать перечисление (например, `пусть diagnoz это перечисление с константами Грипп, Ангина, Пневмония,` тогда поле с диагнозом должно быть описано так: diagnoz bolezn`;)`. Информация в файле должна соответствовать перечислению.

Создать класс-прототип для хранения и обработки экземпляров структур, наложить ограничение на параметр типа данных: элементы коллекции должны быть значимого типа, тип –аргумент должен реализовывать интерфейс *IComparable*.

Написать Windows-приложение, выполняющее следующее функции:

- Считывание данных из файла.
- Вывод информации о пациентах в виде таблицы с продолжительности пребывания в больнице в днях и возраста пациента.
- Добавление данных о новом пациенте
- Сортировку по убыванию продолжительности пребывания в больнице или по возрасту пациента (по выбору пользователя)
- Поиск пациентов не старше 20 лет с заданным диагнозом заболевания (использовать стандартную коллекцию *ArrayLis***t**)

Вариант 8

Подготовить текстовый файл, содержащий информацию о подписчиках на журналы: фамилию, дата рождения, названия журналов (разделителем в файле служит точка с запятой).

Описать структуру. Структура должна содержать следующие элементы: поля для хранения фамилии, адреса, поле-массив с названиями журналов, индексатор для доступа к элементам поля-массива, метод для определения возраста. Для описания последнего поля необходимо создать перечисление с названиями журналов

(например, пусть pressa это перечисление с константами Мурзилка, Наука_и_жизнь, Крокодил, тогда поле-массив должно быть описано так: pressa[] podpiska;). Информация в файле должна соответствовать перечислению.

Создать класс-прототип для хранения и обработки экземпляров структур, наложить ограничение на параметр типа данных: элементы коллекции должны быть значимого типа, тип –аргумент должен реализовывать интерфейс *IComparable.*

Написать Windows-приложение, выполняющее следующее функции:

- Считывание данных из файла.
- Вывод информации о подписчиках в виде таблицы с указаниеми возраста подписчика.
- Добавление данных о новом подписчике
- Сортировку по возрастанию возраста подписчика
- Поиск подписчиков не старше 20 лет, выписывающих заданный (использовать стандартную коллекцию *ArrayList*)

Вариант 9

Подготовить текстовый файл, каждая строка которого содержит название фигуры, цвет и координаты вершин, если квадрат

Создать иерархию классов:

Класс «Геометрические фигуры» должен содержать следующие элементы: поле –цвет, поле статус (квадрат, круг, треугольник и т.д.), метод для получения возраста.

Класс квадратов должен содержать дополнительное поле-массив координат вершин, метод получения площади.

Создать класс-прототип для хранения и обработки экземпляров классов, наложить ограничение на параметр типа данных: элементы коллекции должны быть типа «геометрическая фигура» или производных от него типов, тип –аргумент должен реализовывать интерфейс *IComparable.*

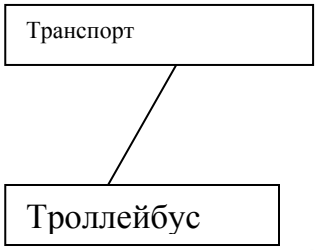
Написать Windows-приложение, выполняющее следующее функции:

- Считывание данных из файла.
- Вывод информации в виде таблицы.
- Сортировку по цвету фигуры
- Формирование и вывод списка квадратов, имеющих площадь не ниже заданной, цвет которых красный (использовать стандартную коллекцию *ArrayList*).

Вариант 10

Подготовить текстовый файл, каждая строка которого содержит название транспорта, год выпуска, регистрационный номер, у троллейбусов дополнительные поля номер маршрута и пробег.

Создать иерархию классов:



Класс «Транспорт» должен содержать следующие элементы: поле *вид транспортного средства* (автомобиль, автобус, трамвай, трактор и т.д.), поле *год выпуска*, поле *регистрационный номер*,

Класс троллейбусов должен содержать дополнительные поля *номер маршрута* и *пробег*, конструктор с параметрами, свойства для чтения полей класса**;**

Создать класс-прототип для хранения и обработки экземпляров классов, наложить ограничение на параметр типа данных: элементы коллекции должны быть типа «транспорт» или производных от него типов, тип –аргумент должен реализовывать интерфейс *IComparable*.

Написать Windows-приложение, выполняющее следующее функции:

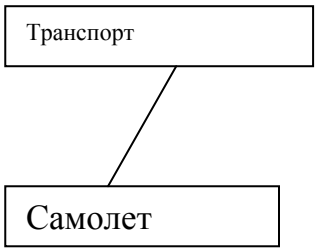
- Считывание данных из файла.
- Вывод информации в виде таблицы.
- Сортировку по году выпуска
- Добавление нового троллейбуса
- Формирование и вывод списка троллейбусов, пробег которых не превышает заданный и совпадает с указанным маршрутом (использовать стандартную коллекцию *ArrayList*).

Вариант 11

Подготовить текстовый файл, каждая строка которого содержит название транспорта, год выпуска, регистрационный номер, у самолетов

дополнительные поля количеством свободных мест; поле – номер рейса; поле – массив, содержащий стоимость билета

Создать иерархию классов:



Класс «Транспорт» должен содержать следующие элементы: поле *вид транспортного средства* (автомобиль, автобус, трамвай, трактор и т.д.), поле *год выпуска*, поле *регистрационный номер*,

Класс самолетов должен содержать поле с количеством свободных мест; поле – номер рейса; пункт назначения, поле – массив, содержащий стоимость билета; индексатор, в котором индекс может принимать значения эконом, бизнес, первый.

Создать класс-прототип для хранения и обработки экземпляров классов, наложить ограничение на параметр типа данных: элементы коллекции должны быть типа «транспорт» или производных от него типов, тип –аргумент должен реализовывать интерфейс IComparable.

Написать Windows-приложение, выполняющее следующее функции:

- Считывание данных из файла.
- Вывод информации в виде таблицы.
- Сортировку по виду транспортного средства
- Добавление нового самолета
- Формирование и вывод списка самолетов, в заданный пункт с ценой билета заданного (использовать стандартную коллекцию *Stack)*.

Вариант 12

Подготовить текстовый файл, каждая строка которого содержит фамилию студента, номер группы, успеваемость (массив)

Описать класс Студент, содержащий поля: фамилия, номер группы, успеваемость (массив), индексатор для доступа к элементам массива оценок, свойство определения количества двоечников.

Создать класс-прототип для хранения и обработки экземпляров классов, наложить ограничение на параметр типа данных: элементы коллекции должны быть типа Студент, тип –аргумент должен реализовывать интерфейс IComparable.

Написать Windows-приложение, выполняющее следующее функции:

- Считывание данных из файла.
- Вывод информации в виде таблицы.
- Добавление данных о студенте.
- Сортировку студентов в алфавитном порядке.
- Формирование и вывод списка двоечников, в заданной группе (использовать стандартную коллекцию *ArrayList*).

Вариант 13

Подготовить текстовый файл, каждая строка которого содержит фамилию сотрудника, наименование отдела, зарплата за последние 6 месяцев (массив), Описать класс Сотрудник, содержащий поля: фамилия, наименование отдела, зарплата за последние 6 месяцев (массив), индексатор для доступа к элементам массива зарплат, свойство определения средней заработной платы.

Создать класс-прототип для хранения и обработки экземпляров классов, наложить ограничение на параметр типа данных: элементы коллекции должны быть типа Сотрудник, тип –аргумент должен реализовывать интерфейс *IComparab*le.

Написать Windows-приложение, выполняющее следующее функции:

- Считывание данных из файла.
- Вывод информации в виде таблицы.
- Добавление данных о сотруднике.
- Сортировку сотрудников в алфавитном порядке или по убыванию средней зарплаты .
- Формирование и вывод списка сотрудников, минимальная зарплата которых меньше заданной (использовать стандартную коллекцию *Stack*).

Вариант 14

Подготовить текстовый файл, каждая строка которого содержит поля: фамилия преподавателя, кафедра, нагрузка в часах 10 месяцев (массив).

Описать класс Преподаватель, содержащий поля: фамилия преподавателя, кафедра, нагрузка в часах 10 месяцев (массив), свойство определения средней заработной платы.

Создать класс-прототип для хранения и обработки экземпляров классов, наложить ограничение на параметр типа данных: элементы коллекции должны быть типа Преподаватель, тип –аргумент должен реализовывать интерфейс *IComparabl*e.

Написать Windows-приложение, выполняющее следующее функции:

- Считывание данных из файла.
- Вывод информации в виде таблицы.
- Сортировку Преподавателей в алфавитном порядке или по возрастанию средней зарплаты.
- Формирование и вывод списка преподавателей, у которых средняя нагрузка превышает среднемесячную по университету (использовать стандартную коллекцию *ArrayList*).

Вариант 15

Подготовить текстовый файл, каждая строка которого содержит поля: фамилия спортсмена, год рождения, результаты 6 соревнований (массив)

Описать класс Спортсмен, содержащий поля: фамилия спортсмена, год рождения, результаты последних 6 соревнований, свойство определения средней результата, определения возраста.

Создать класс-прототип для хранения и обработки экземпляров классов, наложить ограничение на параметр типа данных: элементы коллекции должны быть типа Спортсмен, тип –аргумент должен реализовывать интерфейс *IComparable*.

Написать Windows-приложение, выполняющее следующее функции:

- Считывание данных из файла.
- Вывод информации в виде таблицы.
- Сортировку Спротсменов в алфавитном порядке или по возрастанию средней результата или году рождения .
- Формирование и вывод списка спортсменов, моложе 20 лет, средний результат которых не превышает заданный нагрузка превышает среднемесячную по университету (использовать стандартную коллекцию *HashTable*).

5.3 Вопросы для самоконтроля

1. Работа с коллекциями. Обзор коллекций.
2. Класс ArrayList
3. Класс Hashtable.
4. Класс Stack.
5. Классы прототипы, параметризованные коллекции.
6. Создание класса-прототипа.
7. Обобщенные(параметризованные) методы, ограничения на использование параметризованных типов

Список использованных источников

1. Павловская Т.А. C#. Программирование на языке высокого уровня: Учебник для вузов. – СПб.: Питер, 2007. – 432с.
2. Шилдт Герберт –Полное руководство C# 4.0: Пер. с англ. – М.: ООО «И.Д. Вильяме», 2011. –1056 с.
3. Шилдт Герберт –Полный справочник по C#: Пер. с англ. /Герберт Шилдт –М.: Издательский дом «Вильямс», 2004. –752 с.
4. Троелсен Э. C# и платформа .NET. Библиотека программиста. – СПб.: Питер, 2002. - 800 с.
5. Прайс Д., Гандэрлой М. Visual C# .NET. Полное руководство. – К.: ВЕК+, СПб.: КОРОНА принт, К.: НТИ, М.: Энтроп, 2004. – 960 с.
6. Фролов А.В., Фролов Г.В. Визуальное проектирование приложений C#. – М.: КУДИЦ-ОБРАЗ, 2003. – 512 с.
7. Романькова Т. Л., Коробейникова Е. В. Конструирование программ и языки программирования: пособие по одноименному курсу для студентов технических специальностей дневной формы: пособие по одноименному курсу для студентов технических специальностей дневной формы обучения. – Гомель: ГГТУ, 2010. - 43 с.
8. Романькова Т. Л. Объектно-ориентированное программирование: курс лекций по одноименной дисциплине для студентов специальности 1-40 01 02 "Информационные системы и технологии (по направлениям) дневной формы обучения.–Гомель :ГГТУ, 2010. -97 с.
9. Мурашко, В. С. Объектно-ориентированное программирование : электронный учебно-методический комплекс дисциплины / В. С. Мурашко, Т. Л. Романькова ; кафедра "Информационные технологии". - Гомель : ГГТУ им. П. О. Сухого, 2012. - 1 папка + 1 электрон. опт. Диск
10. Объектно-ориентированное программирование [Электронный ресурс] : лабораторный практикум по одноименной дисциплине для слушателей специальности 1-40 01 73 "Программное обеспечение информационных систем" заочной формы обучения / Т. Л. Романькова ; Министерство образования Республики Беларусь, Учреждение образования "Гомельский государственный технический университет имени П. О. Сухого", Институт повышения квалификации и переподготовки кадров, Кафедра "Информатика". - Гомель : ГГТУ, 2014. - 84 с.
11. Объектно-ориентированное программирование : курс лекций по одноименной дисциплине для слушателей специальности 1-40 01 73 "Программное обеспечение информационных систем" заочной формы обучения / Т. Л. Романькова. - Гомель : ГГТУ, 2013. - 105 с.

Мурашко Валентина Семеновна

РАЗРАБОТКА ПРИЛОЖЕНИЙ С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ

Практикум по дисциплине «Объектно-ориентированное программирование» для слушателей специальности 1-40 01 73 «Программное обеспечение информационных систем» заочной формы обучения

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 03.11.16.

Рег. № 89Е.
http://www.gstu.by