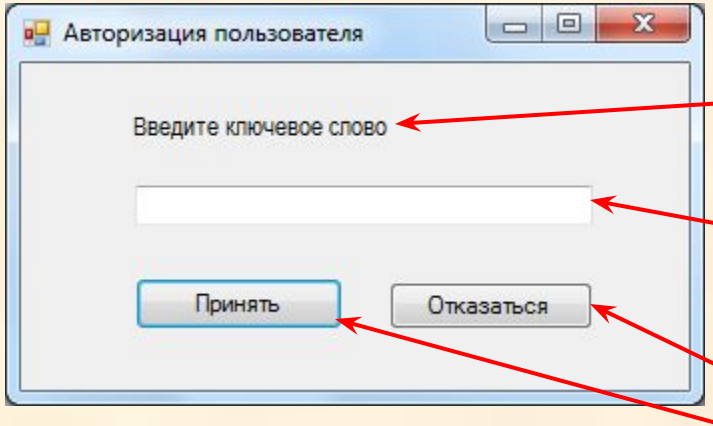
2. Элементы для организации диалога.

Простейшая диалоговая форма содержит визуальные элементы (компоненты), позволяющие выполнять три элементарные операции:

- Вывод текстовой информации на форму (**Label**)
- Ввод текстовой информации пользователем (**TextBox**)
- Выполнение некоторого действия (**Button**)



Вывод текста (**Label**)

Ввод текста (**TextBox**)

Запуск действия (**Button**)

1. Button

Компонент **Button** (кнопка) предназначен для запуска некоторого действия, соответствующего тексту на кнопке.

Главный элемент компонента – событие **Click**.

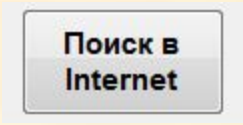
```
private void button1_Click(object sender, EventArgs e)
  {
    <код выполняемого действия>
  }
```

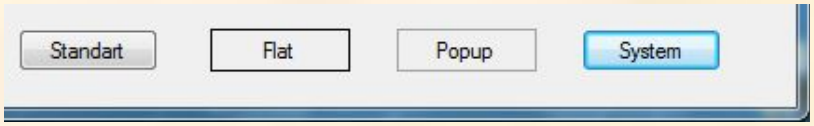
Аргументы обработчика носят уведомительный характер (что случилось, для какого компонента).

Для получения большей информации следует использовать события мыши (**MouseMove**, **MouseDown**, **MouseUp**, **MouseEnter**, **MouseLeave**, **MouseWheel, MouseHover** – задержка мыши на компоненте) или события клавиатуры, которые возникают, если фокус находится на данном компоненте.

Свойства компонента **Button** предназначены для его оформления.

Свойство	*Тип*	*Описание*

Text	string	Надпись на кнопке
		`button1.Text = "Принять";`
		Может содержать несколько строк:
		`button2.Text = "Поиск в\nInternet";`
		 

TextAlign		Выравнивание текста внутри компонента. Значение состоит из двух частей. Первое – по вертикали (Top, Middle, Bottom) Второе – по горизонтали (Left, Center, Right) По умолчанию – MiddleCenter. 
FlatStyle	FlatStyle	Стиль оформления. 
Location	Point	Положение кнопки
Size	Size	Размеры кнопки

Свойство	*Тип*	*Описание*
BackColor	Color	Цвет фона
ForeColor	Color	Цвет текста на кнопке
Font	Font	Оформление текста на кнопке. `button2.Font = new Font("Arial", 11.0F, FontStyle.Bold);`
Enabled	bool	Состояние работоспособности кнопки.
Image	Image	Картинка на поверхности кнопки. `button1.Image = Image.FromFile("D:\\Pict\\AT.BMP");`
ImageAlign		Способ выравнивания изображения (аналогично **TextAlign**). `button1.ImageAlign = ContentAlignment.MiddleLeft;`
Visible	bool	Видимость кнопки.

ToolTip	ToolTip	Подсказка, присоединённая к кнопке.
		`ToolTip tt = new ToolTip();`
		`tt.SetToolTip(button2,"Отмена авторизации");`

2. Label

Компонент **Label** (надпись) предназначен для отображения текста (одно- или многострочного) на форме.

Содержание надписи задаётся свойством **Text**. Для оформления надписи можно использовать свойства **Font**, **Location**, **Size**, **ForeColor**, **BackColor**, **TextAlign**.

Специфические свойства:

Свойство	*Тип*	*Описание*
AutoSize	bool	Если равно **True**, то размер компонента автоматически подстраивается при изменении текста.
MaximumSize **MinimumSize**	Size	Если **AutoSize** равно **True**, то задаёт ограничения для изменения размеров надписи.
BorderStyle		Стиль границы для элемента управления. Граница может быть обычной (**Fixed3D**), тонкой (**FixedSingle**) или отсутствовать (**None**). `Label2.BorderStyle = BorderStyle.None;`

3. TextBox

Компонент **TextBox** (окно ввода текста) предназначен для отображения и изменения одно- или многострочного текста на форме в отдельном прямоугольном окне.

Содержание текста определяется свойством **Text**. Для оформления окна набора текста можно использовать свойства **Font**, **Location**, **Size**, **ForeColor**, **BackColor**, **TextAlign, BorderStyle**.

При этом компонент имеет ряд специфических свойств:

Свойство	*Тип*	*Описание*
MaxLength	int	Число символов, которое можно ввести в элемент управления. Значение по умолчанию равно 32767.

PasswordChar	char	Знак, используемый для маскировки ввода знаков в однострочный элемент управления (символ с кодом 0, если маскировать знаки при вводе в элемент управления не нужно – по умолчанию).
		`textBox1.PasswordChar = '*';`
		`textBox1.PasswordChar = \0;`
		`textBox1.PasswordChar = (char)55; // = '7'`

Свойство	*Тип*	*Описание*
MultiLine	bool	Если **True**, то разрешён ввод многострочного текста.

Lines	string[]	Массив строк, который содержит текст элемента управления в случае **MultiLine** == **True**.
		`string[] temp = textBox2.Lines;`
		`for (int k = 0; k < temp.Length; k++)`
		`{`
		`  temp[k] = Convert.ToString(k+1) + " " + temp[k];`
		`}`
		`textBox2.Lines = temp;`

ReadOnly	bool	Разрешает или запрещает редактирование отображаемого текста.
ScrollBars		Наличие полос прокрутки. **None** – отсутствуют **Horizontal** – т/о снизу **Vertical** – т/о справа **Both** – и снизу и справа.
WordWrap	bool	Если **True**, то при вводе длинного текста курсор автоматически переходит на следующую строчку.

Свойство	*Тип*	*Описание*
SelectedText	string	Выделенный текст.
SelectionLength	int	Длина выделенного текста.
SelectionStart	int	Позиция первого символа выделенного текста во всём тексте. Если нет выделенного текста, то указывает позицию ввода очередного символа.
Dock	DockStyle	Способ привязки компонента к границам свободной клиентской области родителя. **None** – привязка отсутствуют **Top** – привязка к верхней границе (присоединение к верхней границе, ширина компонента совпадает с шириной свободной области родителя) **Bottom, Left, Right** – аналогично **Fill** – заполнение всей свободной области.

Методы компонента **TextBox**:

Метод	*Описание*

AppendText	Добавляет текст в конец содержания **TextBox**.
	`if (textBox1.SelectionLength != 0)`
	`textBox2.AppendText(textBox1.SelectedText);`

Clear()	Удаляет текст
Copy()	Копирует выделенный текст в буфер обмена
Cut()	Вырезает выделенный текст в буфер обмена
Paste()	Заменяет выделенный текст содержимым буфера обмена
SelectAll()	Выделяет весь текст
Undo()	Отменяет последнюю операцию редактирования

События компонента **TextBox**.

Работают все стандартные события визуальных компонент.

Клавиатурные события используются для отслеживания нажимаемых клавиш клавиатуры при вводе текста.

Например, в следующем примере кода ввод символов в **TextBox** ограничен цифрами, запятой и клавишей <BackSpace>:

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (((e.KeyChar < '0') || (e.KeyChar > '9'))
          && (e.KeyChar != ',') && (e.KeyChar != (char)(8) ))
              e.KeyChar='\0';
}
```

Здесь все «неправильные» символы заменяются пустым символом (символом с кодом 0).

Важным для компонента **TextBox** является событие TextChanged, которое генерируется при любом изменении текста в компоненте (независимо от того, происходит это пользователем или программно).

Параметры события имеют уведомительный характер, поэтому обработчик может только анализировать состояние текста, которое уже изменилось.

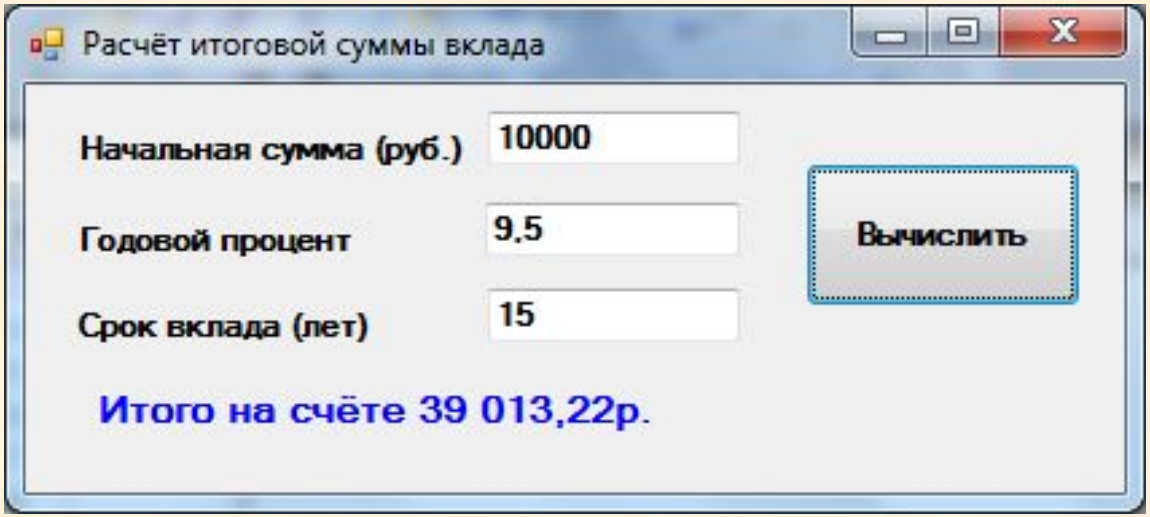
В примере показано изменение свойства **ForeColor** объекта **TextBox**. В примере текст преобразуется в десятичное число, а свойство **ForeColor** изменяется на **Color.Red**, если число является отрицательным, и на **Color.Green**, если число положительное.

```
private void TextBox1_TextChanged(object sender, EventArgs e)
{
  try
  { // текст приводится к типу double и проверяется на отрицательность.
      if (double.Parse(textBox1.Text) < 0)
           { textBox1.ForeColor = Color.Red; }
      else { textBox1.ForeColor = Color.Green; }
  }
  catch
  { // если в тексте есть ошибка, то используется системный цвет
    textBox1.ForeColor = SystemColors.ControlText;
  }
}
```

7. Пример расчётной задачи.

Пусть требуется написать программу для расчёта суммы вклада, если известны:

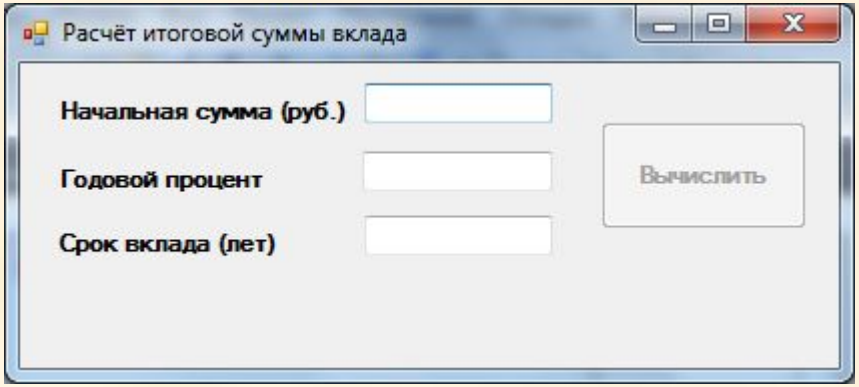
- Начальная сумма вклада
- Процент вклада
- Срок вклада



Решение

Расположим на форме следующие компоненты:

- Четыре компонента **Label**, три из которых будут отвечать за пояснения к исходным данным, а четвёртый **label4** – за вывод результатов расчёта.
- Три компонента **TextBox**, отвечающие за ввод исходных значений (по порядку).
- Один компонент **Button**, запускающий процедуру расчёта итоговой суммы вклада.



Настроим компоненты так, как показано на рисунке, установив шрифт для всех в состояние **Bold**.

У компонента **label4** увеличим шрифт и изменим цвет символов **ForeColor**.

У **button1** установим в значение **false** свойство **Enabled**.

Реализуем обработку ввода исходных данных.

Для компонентов **textBox1**, **textBox2** и **textBox3** запишем единый обработчик события **KeyPress** (нажатие символьной клавиши):

```
private void textBox3_KeyPress(object sender,
                               KeyPressEventArgs e)
{
  if ((e.KeyChar >= '0') && (e.KeyChar <= '9')) return;
  if (e.KeyChar == (char)8) return;
  if (e.KeyChar == '.') e.KeyChar = ',';
  ...
```

Здесь первой командой пропускаются (разрешаются) символы – цифры.
Второй командой разрешается символ с кодом 8 – <BackSpace> (забой).
Третьей командой символ <точка> (английский десятичный разделитель) заменяется на символ <запятая> (как принято в России).

Далее:

```
...
   if (e.KeyChar == ',')
   {
      if ((((TextBox)sender).Text.IndexOf(',') != -1)
                  || (((TextBox)sender).Text.Length == 0))
            e.Handled = true;
      return;
    }
...
```

Если введён десятичный разделитель – символ <запятая>, то в случае присутствия в **TextBox** ещё одной запятой, или если **TextBox** пустой, то ввод символа отменяется.

Обратите внимание, что данный обработчик – общий для всех трёх окошек ввода данных. Поэтому, чтобы обратиться к тому компоненту, который был в фокусе, использован параметр **sender**, а чтобы получить доступ к его свойствам, использовано приведение типа **(TextBox)sender**.

Далее:

```
...
   if (e.KeyChar == (char)Keys.Enter)
   {
      if (sender.Equals(textBox1)) textBox2.Focus();
      if (sender.Equals(textBox2)) textBox3.Focus();
      if (sender.Equals(textBox3)) button1.Focus();
   }
   e.Handled = true;
}
```

Если была нажата клавиша <ENTER>, то фокус перемещается на следующий компонент: с **textBox1** на **textBox2**, с **textBox2** на **textBox3**, с **textBox3** на **button1** – кнопку «Вычислить».

В конце обработчика введённый символ аннулируется. Поэтому, для обработки операционной системой останутся те символы, которые были пропущены ранее (**return** – принудительное завершение функции).

Реализуем обработку события – изменение полей ввода исходных данных.

Для компонентов **textBox1**, **textBox2** и **textBox3** запишем единый обработчик события **TextChanged** (изменение текста):

```
private void textBox3_TextChanged(object sender,
                                  EventArgs e)
{
   label4.Text = "";
   if ((textBox1.Text == "") || (textBox2.Text == "")
                             || (textBox3.Text == ""))
                button1.Enabled = false;
            else button1.Enabled = true;
}
```

Здесь вначале очищается поле вывода результатов **label4**.
Затем, в случае, когда хотя бы одно из полей ввода исходных данных пусто, кнопка «Вычислить» отключается. Иначе – она вполне работоспособна.

Реализуем главное действие программы – нажатие кнопки «Вычислить».

Для компонента **button1** запишем обработчик события **Click** (нажатие на кнопку, происходит при щелчке мышкой или клавишей <ENTER>, когда эта кнопка находится в фокусе):

```
private void button1_Click(object sender, EventArgs e)
{
   double start = Convert.ToDouble(textBox1.Text);
   double proc  = Convert.ToDouble(textBox2.Text);
   double srok  = Convert.ToDouble(textBox3.Text);
   double fin   = start * Math.Pow(1 + proc/100, srok);
   label4.Text = "Итого на счёте "+fin.ToString("C");
}
```

Здесь локальным переменным `start, proc, srok` присвоены числовые значения исходных данных, введённые в текстовом формате.
Локальной переменной `fin` присвоено значение результата. При этом использована функция `Pow` из класса `Math`, вычисляющая степень числа.
Для вывода ответа значение переменной `fin` преобразовано в текстовый вид с использованием формата денежных единиц (параметр `"C"`).

8. Сохранение строк в текстовом файле.

Файловый ввод вывод данных в C# организуется через потоки – механизм Windows, формирующий наборы данных произвольной длины, для которых реализованы функции добавления и извлечения данных.

Файл рассматривается как поток данных, оформленный на внешнем носителе информации.

Для доступа к файлу следует создать переменную класса **FileStream**, связав её с конкретным файлом, например, так:

```
FileStream fs = new FileStream(<имя файла>, <режим>);
```

Здесь `<имя файла>` – полное или краткое имя файла,

`<режим>` – способ открытия файла. Имеет варианты:

Create – файл создаётся заново

Append – файл открывается для дозаписи

Open – файл открывается, режим доступа FileAccess (третий параметр конструктора `FileStream()`.

Для записи в файл используется объект класса **StreamWriter**, при создании которого указывается соответствующий файловый поток:

```
StreamWriter sw = new StreamWriter(<файловый поток>);
```

Данный объект представляет группу методов, обслуживающих запись в файл:

Метод	*Описание*
Write(<значение>)	Сохраняет текстовое представление указанного значение в файл. Допускаются типы: булевый, символьный, целый, вещественный, строковый.
WriteLine(<значение>)	Аналог **Write(<значение>)**, после вывода значения записывается признак конца строки.
WriteLine()	Просто записывается признак конца строки.
Flush()	Освобождается буфер вывода.
Close()	Закрывает файловый поток.

Классы **FileStream** и **StreamWriter** описаны в пространстве имён **System.IO**, которое необходимо предварительно заказать:

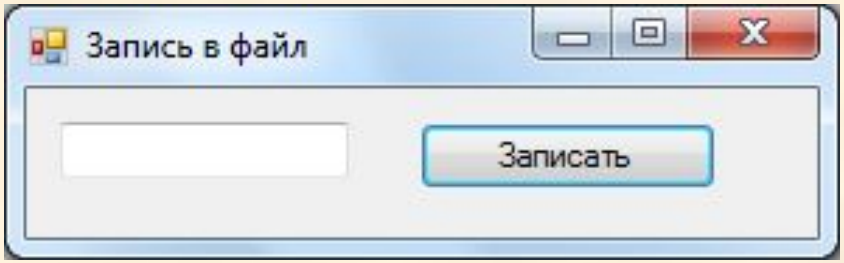
```
using System.IO;
```

Пример.

Записываем в текстовый файл “a.txt” введённое число и его удвоенное значение. Каждый вывод предваряется информацией о текущих дате и времени:

```
----------------------------------------
Hello world
this is 25.10.2011 22:10:43
Текущее = 12
Квадрат = 144
```

Форма для ввода данных имеет вид:



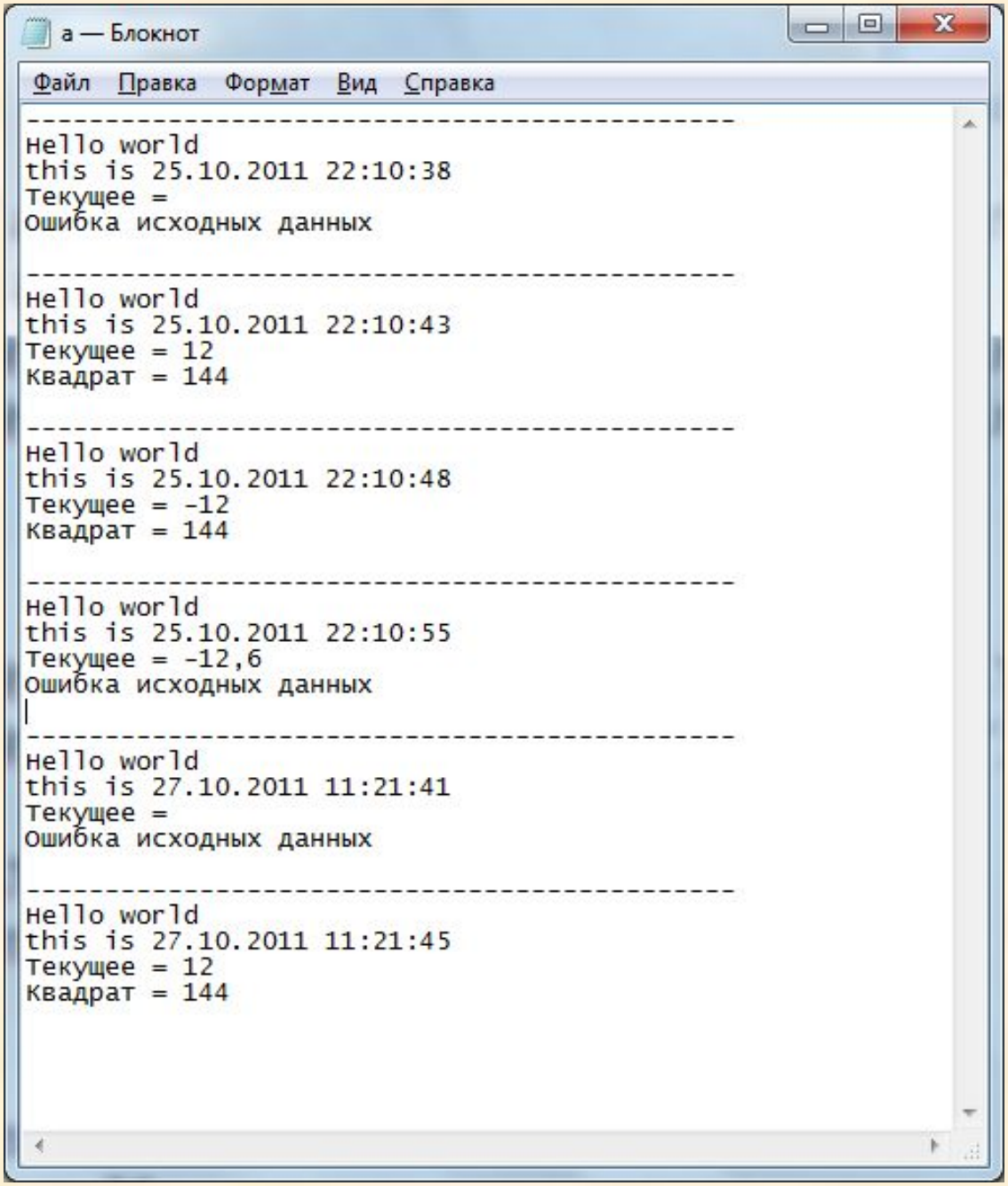
Обработчик события **Click** для **Button**:


```
private void button3_Click(object sender, EventArgs e)
{
   FileStream fs;
   DateTime dt = DateTime.Now;
   string s1 = dt.ToShortDateString();
   string s2 = dt.ToLongTimeString();
   try
      { fs = new FileStream("a.txt", FileMode.Append); }
   catch
      { fs = new FileStream("a.txt", FileMode.Create); }
   StreamWriter sw = new StreamWriter(fs);
   sw.WriteLine("-------------------------------------");
   sw.WriteLine("Hello world");
   sw.WriteLine("this is " +  s1 + " " + s2);
   ...
```


```
    ...
    try
    {
      int a1 = Convert.ToInt32(textBox1.Text);
      int a2 = a1 * a1;
      sw.WriteLine("Текущее = " + a1.ToString());
      sw.WriteLine("Квадрат = " + a2.ToString());
    }
    catch
    {
      sw.WriteLine("Текущее = " + textBox1.Text);
      sw.WriteLine("Ошибка исходных данных");
    }
    sw.WriteLine();
    sw.Close();
  }
```

Предварительно заказано

```
using System.IO;
```


а — Блокнот
Файл Правка Формат Вид Справка

Hello world
this is 25.10.2011 22:10:38
Текущее =
Ошибка исходных данных

Hello world
this is 25.10.2011 22:10:43
Текущее = 12
Квадрат = 144

Hello world
this is 25.10.2011 22:10:48
Текущее = -12
Квадрат = 144

Hello world
this is 25.10.2011 22:10:55
Текущее = -12,6
Ошибка исходных данных

Hello world
this is 27.10.2011 11:21:41
Текущее =
Ошибка исходных данных

Hello world
this is 27.10.2011 11:21:45
Текущее = 12
Квадрат = 144