

Практическая работа №1

Создание приложения с использованием WindowsForms

Цель: научиться разрабатывать приложения, работать с формами и элементами интерфейса. Изучить свойства форм.

Процесс разработки программы в Microsoft Visual C# рассмотрим на примере — создадим *приложение* (программы, предназначенные для решения прикладных задач, принято называть приложениями), позволяющее посчитать доход по вкладу (рис. 2.1).

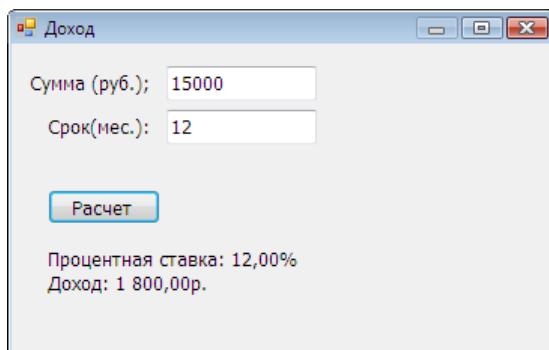


Рис. 2.1. Окно программы "Доход"

Начало работы над проектом

Чтобы начать работу над новым проектом, надо:

1. В меню **File** выбрать команду **New Project**.
2. В открывшемся окне **New Project** выбрать тип приложения — **Windows Forms Application - Visual C#**.
3. В поле **Name** ввести имя проекта — **profit** и нажать кнопку **OK** (рис. 2.2).

В результате описанных действий в папке временных проектов (по умолчанию это C:\Users\User\AppData\Local\Temporary Projects) будет создана папка profit, а в ней — проект profit.

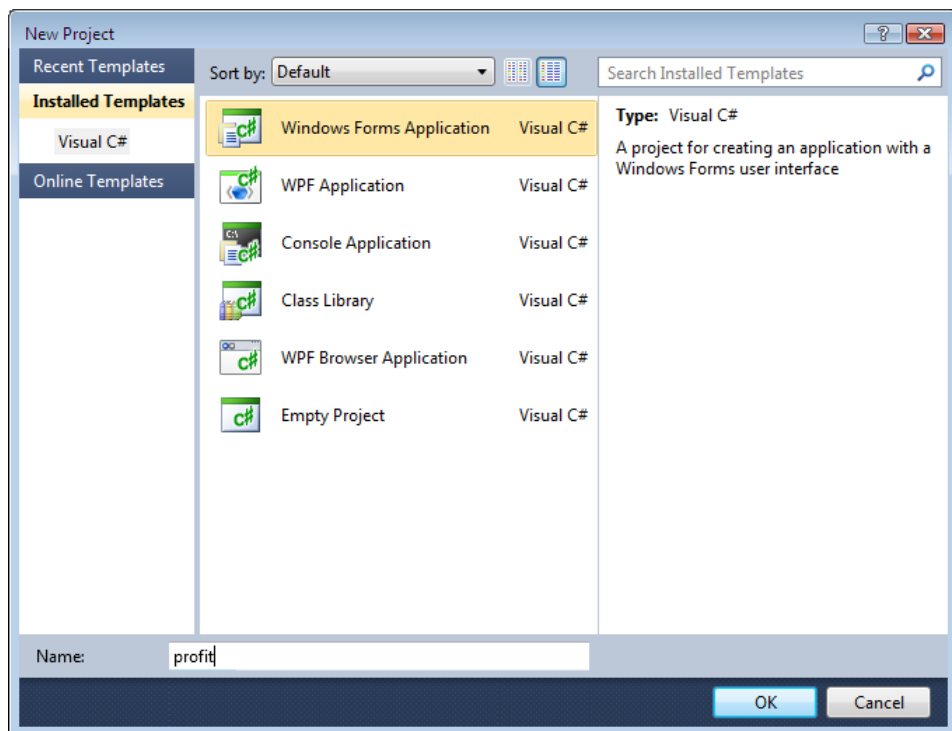


Рис. 2.2. Начало работы над новой программой

Форма

Работа над приложением начинается с создания стартовой *формы* — главного окна программы. Форма создается путем добавления на заготовку формы необходимых компонентов и изменения значений свойств самой формы.

Сначала нужно установить требуемые значения свойств формы, затем — поместить на форму необходимые *компоненты* (поля ввода информации, командные кнопки, поля отображения текста и др.) и выполнить.

Настройка формы (а также компонентов) осуществляется путем изменения значений *свойств*. Свойства *объекта* (формы, компонента) определяют его вид и поведение. Например, свойство `Text` определяет текст заголовка окна, а свойство `StartPosition` — положение окна в момент появления его на экране.

Основные свойства формы (объекта `Form`) приведены в табл. 2.1.

Таблица 2.1. Свойства формы (объекта `Form`)

Свойство	Описание
Name	Имя формы
Text	Текст в заголовке

Таблица 2.1 (окончание)

Свойство	Описание
Size	Размер формы. Уточняющее свойство Width определяет ширину, свойство Height — высоту
StartPosition	Положение формы в момент первого появления на экране. Форма может находиться в центре экрана (CenterScreen), в центре родительского окна (CenterParent). Если значение свойства равно Manual, то положение формы определяется значением свойства Location
Location	Положение формы на экране. Расстояние от верхней границы формы до верхней границы экрана задает уточняющее свойство Y, расстояние от левой границы формы до левой границы экрана — уточняющее свойство X
FormBorderStyle	Тип формы (границы). Форма может представлять собой обычное окно (Sizable), окно фиксированного размера (FixedSingle, Fixed3D), диалог (FixedDialog) или окно без кнопок Свернуть и Развернуть (SizeableToolWindow, FixedToolWindow). Если свойству присвоить значение None, у окна не будет заголовка и границы
ControlBox	Управляет отображением системного меню и кнопок управления окном. Если значение свойства равно False, то в заголовке окна кнопка системного меню, а также кнопки Свернуть , Развернуть , Закрыть не отображаются
MaximizeBox	Кнопка Развернуть . Если значение свойства равно False, то находящаяся в заголовке окна кнопка Развернуть недоступна
MinimizeBox	Кнопка Свернуть . Если значение свойства равно False, то находящаяся в заголовке окна кнопка Свернуть недоступна
Icon	Значок в заголовке окна
Font	Шрифт, используемый по умолчанию компонентами, находящимися на поверхности формы. Изменение значения свойства приводит к автоматическому изменению значения свойства Font всех компонентов формы (при условии, что значение свойства компонента не было задано явно)
ForeColor	Цвет, наследуемый компонентами формы и используемый ими для отображения текста. Изменение значения свойства приводит к автоматическому изменению соответствующего свойства всех компонентов формы (при условии, что значение свойства Font компонента не было задано явно)
BackColor	Цвет фона. Можно задать явно (выбрать на вкладке Custom или Web) или указать элемент цветовой схемы (выбрать на вкладке System)
Opacity	Степень прозрачности формы. Форма может быть непрозрачной (100%) или прозрачной. Если значение свойства меньше 100%, то сквозь форму видна поверхность, на которой она отображается

Для изменения значений свойств объектов используется окно **Properties**. В левой колонке окна перечислены свойства объекта, *выбранного* в данный момент, в правой — указаны значения свойств. Имя выбранного объекта отображается в верхней части окна **Properties**.

Чтобы в заголовке окна отображалось название программы, надо изменить значение свойства `Text`. Для этого следует щелкнуть левой кнопкой мыши в поле значение свойства `Text` (в поле появится курсор), ввести в поле редактирования текст **Доход** и нажать клавишу `<Enter>` (рис. 2.3).

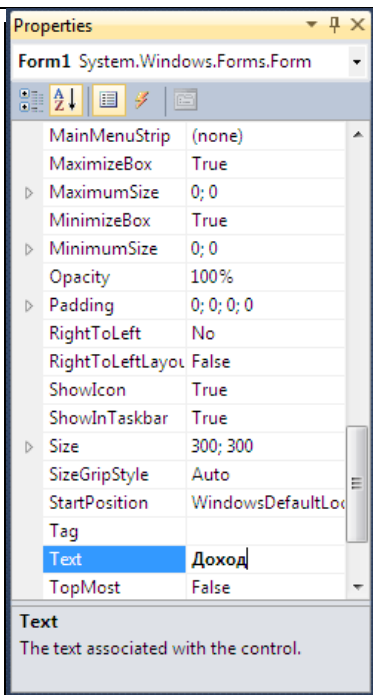


Рис. 2.3. Изменение значения свойства `Text` путем ввода нового значения

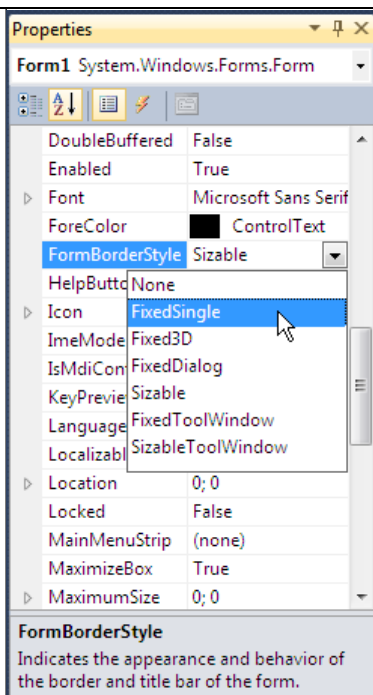


Рис. 2.4. Установка значения свойства путем выбора из списка

При выборе некоторых свойств, например `FormBorderStyle`, справа от текущего значения свойства появляется значок раскрывающегося списка. Очевидно, что значение таких свойств можно задать путем выбора из списка (рис. 2.4).

Некоторые свойства являются сложными. Они представляют собой совокупность других (уточняющих) свойств. Например, свойство `Size`, определяющее размер формы, представляет собой совокупность свойств `Width` и `Height`. Перед именами сложных свойств стоит значок **▶**, в результате щелчка на котором раскрывается список уточняющих свойств (рис. 2.5). Значение уточняющего свойства можно задать (изменить) обычным образом — ввести нужное значение в поле редактирования.

Размер формы можно изменить и с помощью мыши, точно так же, как и любого окна, т. е. путем перемещения границы. По окончании перемещения границы значения свойств `Width` и `Height` будут соответствовать установленному размеру формы.

В результате выбора некоторых свойств, например `Font`, в поле значения свойства отображается кнопка, на которой изображены три точки. Это значит, что задать

значение свойства можно в дополнительном диалоговом окне, которое появится в результате щелчка на этой кнопке. Например, значение свойства Font можно задать путем ввода значений уточняющих свойств (Name, Size, Style и др.), а можно воспользоваться стандартным диалоговым окном **Шрифт**, которое появится в результате щелчка на кнопке с тремя точками (рис. 2.6).

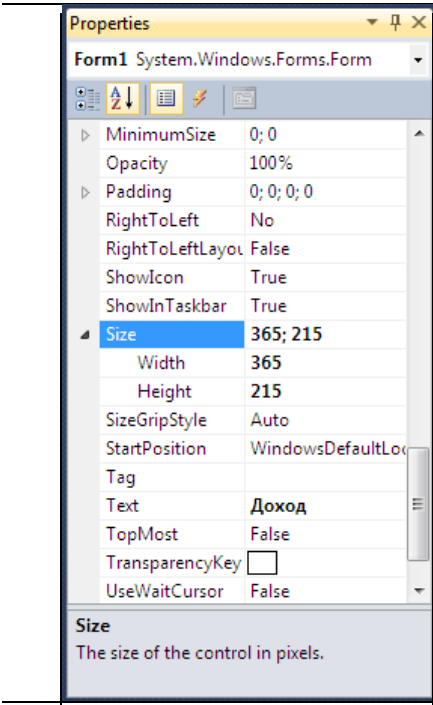


Рис. 2.5. Изменение значения уточняющего свойства

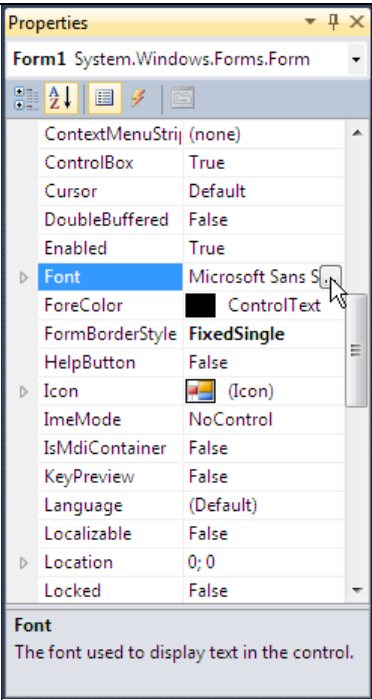


Рис. 2.6. Чтобы задать свойства шрифта, щелкните на кнопке с тремя точками

В табл. 2.2 приведены значения свойств формы программы "Доход". Значения остальных свойств формы оставлены без изменения и поэтому в таблице не представлены. Обратите внимание, в именах некоторых свойств есть точка. Это значит, что это значение уточняющего свойства.

Таблица 2.2. Значения свойств стартовой формы

Свойство	Значение	Комментарий
Text	Доход	
Size.Width	365	
Size.Height	215	
FormBorderStyle	FixedSingle	Тонкая граница формы. Во время работы программы пользователь не сможет изменить размер окна путем захвата и перемещения его границы

Таблица 2.2 (окончание)

Свойство	Значение	Комментарий
StartPosition	CenterScreen	Окно программы появится в центре экрана
MaximizeBox	False	В заголовке окна не отображать кнопку Развернуть
Font.Name	Tahoma	
Font.Size	10	

После того как будут установлены значения свойств формы, она должна выглядеть так, как показано на рис. 2.7. Теперь на форму надо добавить *компоненты*.

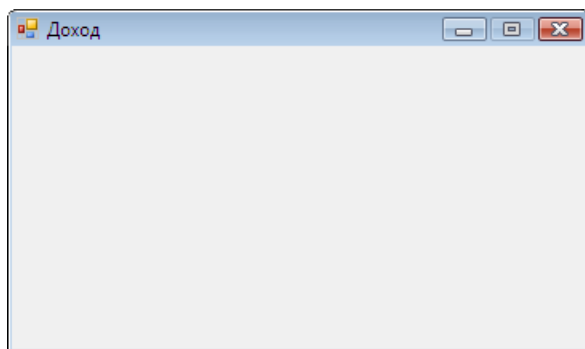


Рис. 2.7. Форма после изменения значений ее свойств

Компоненты

Поля ввода/редактирования, поля отображения текста, командные кнопки, списки, переключатели и другие элементы, обеспечивающие взаимодействие пользователя с программой, называют *компонентами пользовательского интерфейса*. Они находятся в окне **Toolbox** на вкладке **Common Controls**.

Программа "Доход" должна получить от пользователя исходные данные — сумму и срок вклада. Ввод данных с клавиатуры обеспечивает компонент `TextBox`. Таким образом, на форму разрабатываемого приложения нужно поместить два компонента `TextBox`.

Чтобы на форму добавить компонент `TextBox`, надо:

1. В палитре компонентов (окно **Toolbox**) раскрыть вкладку **Common Controls**.
2. Сделать щелчок на значке компонента `TextBox` (рис. 2.8).
3. Установить указатель мыши в ту точку формы, в которой должен быть левый верхний угол компонента, и сделать щелчок левой кнопкой мыши.

В результате на форме появляется поле ввода/редактирования — компонент `TextBox` (рис. 2.9).

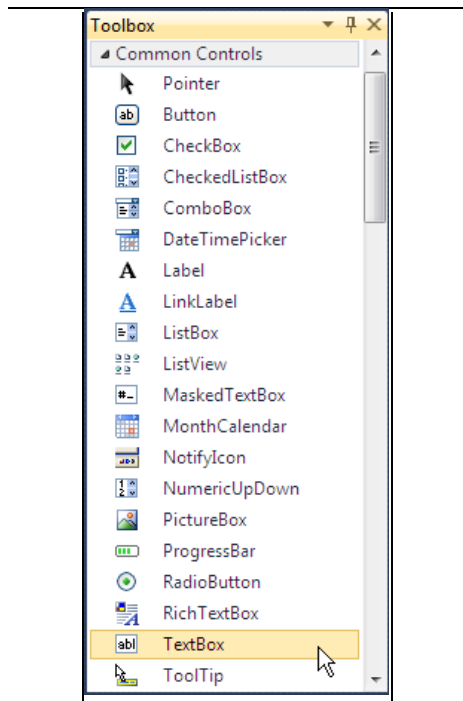


Рис. 2.8. Выбор компонента в палитре
(компонент TextBox – поле редактирования)

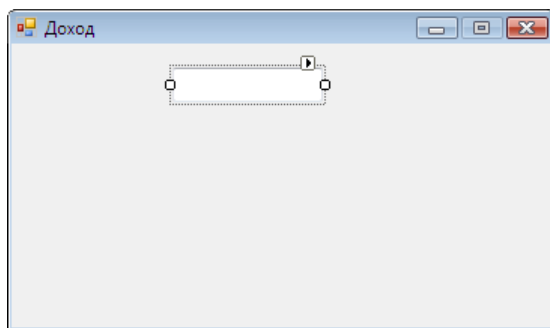


Рис. 2.9. Результат добавления
на форму компонента TextBox

Каждому добавленному компоненту среда разработки присваивает имя, которое состоит из названия компонента и его порядкового номера. Например, первый добавленный на форму компонент TextBox получает имя `textBox1`, второй — `textBox2`. Программист путем изменения значения свойства `Name` может поменять имя компонента. Однако в простых программах имена компонентов, как правило, не меняют.

Основные свойства компонента TextBox приведены в табл. 2.3.

Таблица 2.3. Свойства компонента TextBox

Свойство	Описание
Name	Имя компонента. Используется для доступа к компоненту и его свойствам
Text	Текст, который находится в поле редактирования
Location	Положение компонента на поверхности формы
Size	Размер компонента
Font	Шрифт, используемый для отображения текста в поле компонента
ForeColor	Цвет текста, находящегося в поле компонента
BackColor	Цвет фона поля компонента
BorderStyle	Вид рамки (границы) компонента. Граница компонента может быть обычной (Fixed3D), тонкой (FixedSingle) или отсутствовать (None)

Таблица 2.3 (окончание)

Свойство	Описание
TextAlign	Способ выравнивания текста в поле компонента. Текст в поле компонента может быть прижат к левой границе компонента (Left), правой (Right) или находиться по центру (Center)
MaxLength	Максимальное количество символов, которое можно ввести в поле компонента
Multiline	Разрешает (True) или запрещает (False) ввод нескольких строк текста
ReadOnly	Разрешает (True) или запрещает (False) редактирование отображаемого текста
Lines	Массив строк, элементы которого содержат текст, находящийся в поле редактирования, если компонент находится в режиме MultiLine. Доступ к строке осуществляется по номеру. Строки нумеруются с нуля
ScrollBars	Задаёт отображаемые полосы прокрутки: Horizontal – горизонтальная; Vertical – вертикальная; Both – горизонтальная и вертикальная; None – не отображать

На рис. 2.10 приведен вид формы программы "Доход" после добавления двух полей ввода/редактирования. Один из компонентов *выбран* — выделен рамкой. Свойства именно этого (выбранного) компонента отображаются в окне **Properties**. Чтобы увидеть и, если надо, изменить свойства другого компонента, нужно этот компонент выбрать — щелкнуть левой кнопкой мыши на изображении компонента в форме или выбрать его имя в раскрывающемся списке, который находится в верхней части окна **Properties** (рис. 2.11).

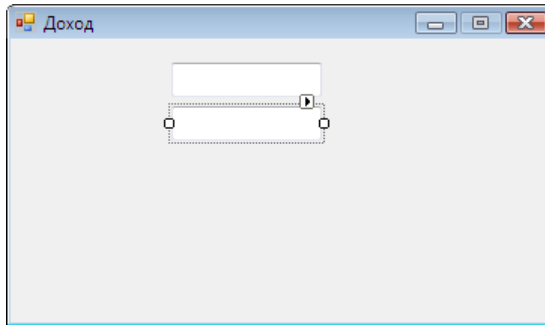


Рис. 2.10. Форма с двумя компонентами TextBox

Значения свойств компонента, определяющих размер и положение компонента на поверхности формы, можно изменить с помощью мыши.

Чтобы изменить положение компонента, необходимо установить курсор мыши на его изображение, нажать левую кнопку мыши и, удерживая ее нажатой, переместить компонент в нужную точку формы (рис. 2.12).

Для того чтобы изменить размер компонента, необходимо сделать щелчок на его изображении (в результате чего компонент будет выделен), установить указатель

мыши на один из маркеров, помечающих границу компонента, нажать левую кнопку мыши и, удерживая ее нажатой, изменить положение границы компонента (рис. 2.13).

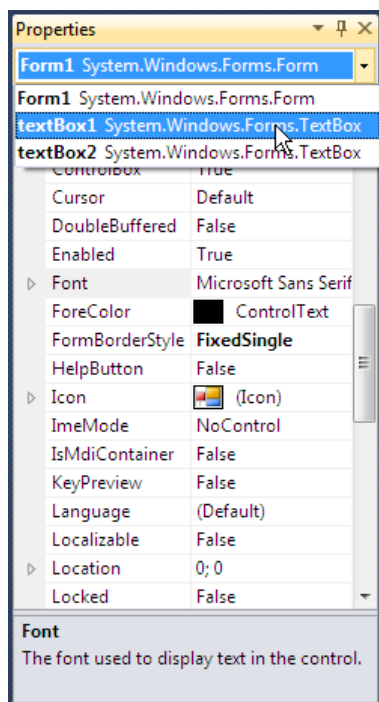


Рис. 2.11. Выбор компонента в окне **Properties**

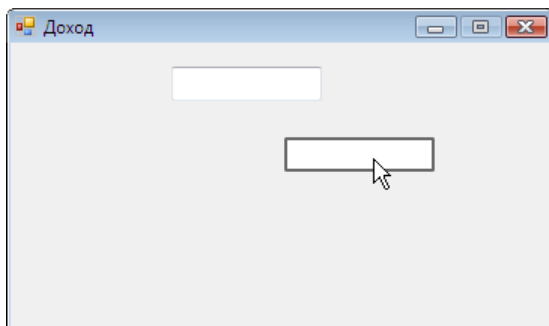


Рис. 2.12. Изменение положения компонента

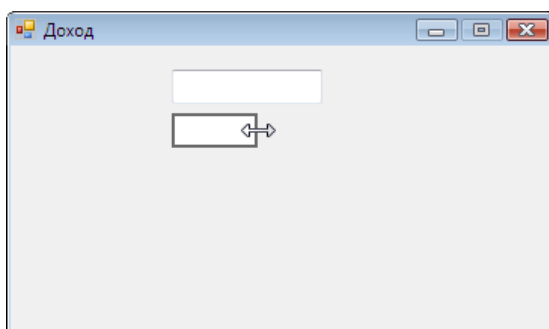


Рис. 2.13. Изменение размера компонента

В табл. 2.4 приведены значения свойств компонентов `textBox1` и `textBox2` (прочёрк показывает, что значением свойства `Text` является пустая строка). Значения остальных свойств компонентов оставлены без изменения и поэтому в таблице не показаны. Компонент `textBox1` предназначен для ввода суммы вклада, `textBox2` — срока. Так как значения свойства `Font` компонентов `TextBox` не были изменены, то во время работы программы текст в полях редактирования будет отображаться шрифтом, заданным для формы. Компоненты `TextBox`, как и другие компоненты, находящиеся на форме, наследуют значение свойства `Font` формы (если значение свойства `Font` компонента не было задано явно). Поэтому если изменить значение свойства `Font` формы, автоматически изменятся значения свойств `Font` компонентов, находящихся на форме. Если требуется, чтобы текст в поле компонента отображался другим шрифтом, нужно явно задать значение свойства `Font` этого компонента.

Форма программы "Доход" после настройки компонентов `TextBox` приведена на рис. 2.14.

Таблица 2.4. Значения свойств компонентов *TextBox*

Компонент	Свойство	Значение
textBox1	Location.X	107
	Location.Y	16
	Size.Width	100
	Size.Height	23
	Text	–
	TabIndex	0
textBox2	Location.X	107
	Location.Y	45
	Size.Width	57
	Size.Height	23
	Text	–
	TabIndex	1

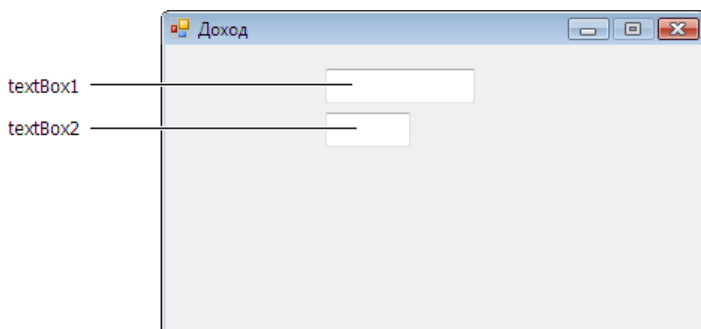


Рис. 2.14. Форма после настройки компонентов *TextBox*

Отображение текста на поверхности формы (подсказок, результата расчета) обеспечивает компонент *Label*. В окне программы "Доход" текст отображается слева от полей ввода/редактирования (информация о назначении полей). Результат расчета также отображается в окне программы. Поэтому в форму надо добавить три компонента *Label* (рис. 2.15).

Добавляется компонент *Label* на форму точно так же, как и поле редактирования (компонент *TextBox*).

Основные свойства компонента *Label* приведены в табл. 2.5.

На форму разрабатываемого приложения надо добавить три компонента *Label*. В полях *label1* и *label2* отображается информация о назначении полей ввода, поле *label3* используется для вывода результата расчета.

Значения свойств компонентов *Label* приведены в табл. 2.6.

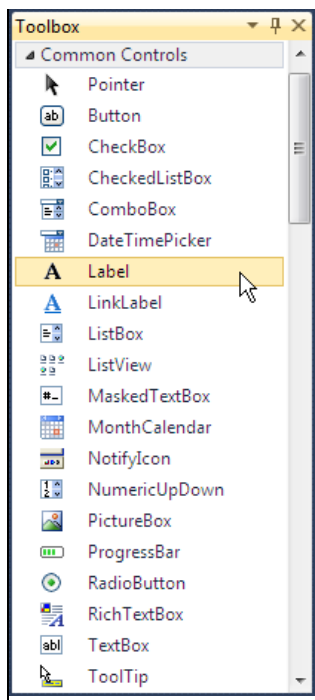


Рис. 2.15. Компонент `Label` – поле отображения текста

Таблица 2.5. Свойства компонента `Label`

Свойство	Описание
<code>Name</code>	Имя компонента. Используется в программе для доступа к свойствам компонента
<code>Text</code>	Отображаемый текст
<code>Location</code>	Положение компонента на поверхности формы
<code>AutoSize</code>	Признак автоматического изменения размера компонента. Если значение свойства равно <code>True</code> , то при изменении значения свойства <code>Text</code> (или <code>Font</code>) автоматически изменяется размер компонента
<code>Size</code>	Размер компонента (области отображения текста). Определяет (если значение свойства <code>AutoSize</code> равно <code>False</code>) размер компонента (области отображения текста)
<code>Font</code>	Шрифт, используемый для отображения текста
<code>ForeColor</code>	Цвет текста, отображаемого в поле компонента
<code>BackColor</code>	Цвет закрашки области вывода текста
<code>TextAlign</code>	Способ выравнивания (расположения) текста в поле компонента. Всего существует девять способов расположения текста. На практике наиболее часто используют выравнивание по левой верхней границе (<code>TopLeft</code>), посередине (<code>TopCenter</code>) и по центру (<code>MiddleCenter</code>)

Таблица 2.6. Значения свойств компонентов *Label*

Компонент	Свойство	Значение
label1	Location.X	13
	Location.Y	19
	AutoSize	False
	Size.Width	88
	Size.Height	20
	Text	Сумма (руб.):
	TextAlign	MiddleRight
label2	Location.X	13
	Location.Y	45
	AutoSize	False
	Size.Width	88
	Size.Height	20
	Text	Срок (мес.):
	TextAlign	MiddleRight
label3	Location.X	23
	Location.Y	122
	AutoSize	False
	Size.Width	299
	Size.Height	50
	Text	-

После настройки компонентов *Label* форма разрабатываемого приложения должна выглядеть так, как показано на рис. 2.16.

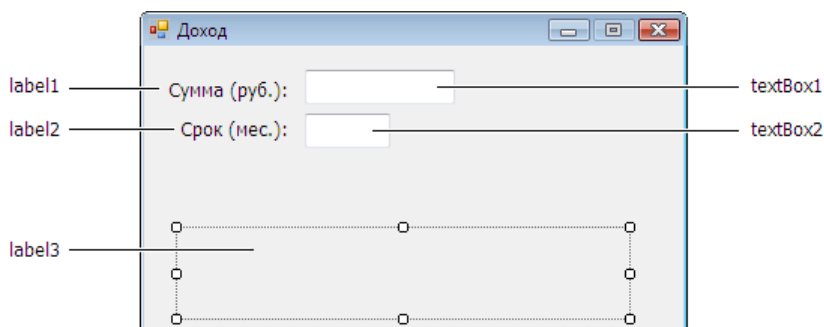


Рис. 2.16. Вид формы после настройки полей отображения текста

Последнее, что надо сделать на этапе создания формы, — добавить на форму командную кнопку **Расчет**. Назначение этой кнопки очевидно.

Командная кнопка, компонент `Button` (рис. 2.17), добавляется на форму точно так же, как и другие компоненты. Значок компонента `Button` находится на вкладке **Common Controls**. Основные свойства компонента `Button` приведены в табл. 2.7.

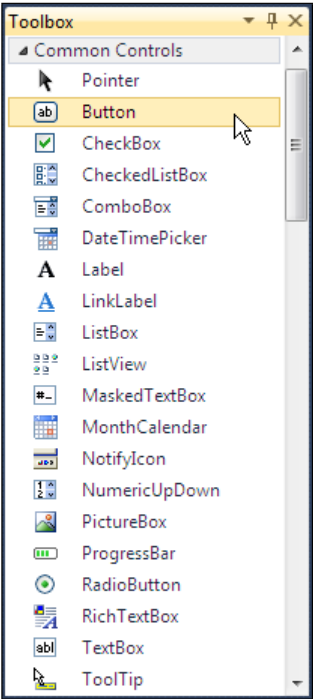


Рис. 2.17. Командная кнопка — компонент `Button`

Таблица 2.7. Свойства компонента `Button`

Свойство	Описание
Name	Имя компонента. Используется для доступа к компоненту и его свойствам
Text	Текст на кнопке
TextAlign	Положение текста на кнопке. Текст может располагаться в центре кнопки (MiddleCenter), быть прижат к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения надписи (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)
FlatStyle	Стиль. Кнопка может быть стандартной (Standard), плоской (Flat) или "всплывающей" (Popup)
Location	Положение кнопки на поверхности формы. Уточняющее свойство X определяет расстояние от левой границы кнопки до левой границы формы, уточняющее свойство Y — от верхней границы кнопки до верхней границы клиентской области формы (нижней границы заголовка)
Size	Размер кнопки
Enabled	Признак доступности кнопки. Кнопка доступна, если значение свойства равно True, и недоступна, если значение свойства равно False (в этом случае нажать кнопку нельзя, событие Click в результате щелчка на ней не возникает)
Visible	Позволяет скрыть кнопку (False) или сделать ее видимой (True)
Cursor	Вид указателя мыши при позиционировании указателя на кнопке

Таблица 2.7 (окончание)

Свойство	Описание
Image	Картинка на поверхности кнопки. Рекомендуется использовать gif-файл, в котором определен прозрачный цвет
ImageAlign	Положение картинки на кнопке. Картинка может располагаться в центре (MiddleCenter), быть прижата к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения картинки на кнопке (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)
ImageList	Набор изображений, из которых может быть выбрано то, которое будет отображаться на поверхности кнопки. Представляет собой объект типа ImageList. Чтобы задать значение свойства, в форму приложения нужно добавить компонент ImageList
ImageIndex	Номер (индекс) изображения из набора ImageList, которое отображается на кнопке
ToolTip	Подсказка, появляющаяся рядом с указателем мыши при его позиционировании на кнопке. Чтобы свойство стало доступно, в форму приложения нужно добавить компонент ToolTip

После того как на форму будут добавлены кнопки, нужно выполнить их настройку. Значения свойств компонентов `Button` приведены в табл. 2.8, окончательный вид формы показан на рис. 2.18.

Таблица 2.8. Значения свойств компонента `button1`

Свойство	Значение
Location.X	26
Location.Y	84
Size.Width	75
Size.Height	23
Text	Расчет

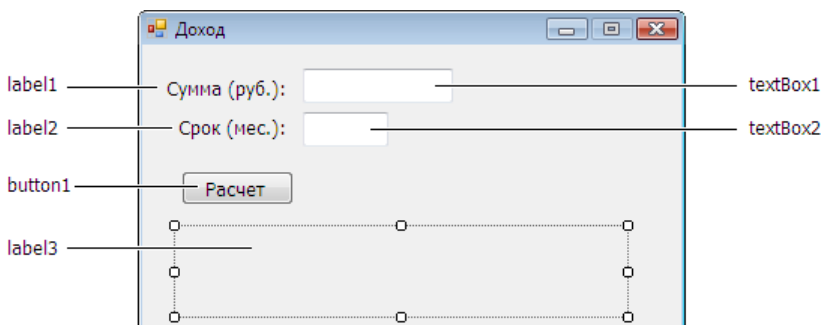


Рис. 2.18. Окончательный вид формы программы "Доход"

Завершив работу по созданию формы, можно приступить к программированию — созданию процедур обработки событий.

Событие

Вид формы программы "Доход" подсказывает, как работает программа. Очевидно, что пользователь должен ввести в поля редактирования исходные данные и сделать щелчок на кнопке **Расчет**. Щелчок на изображении командной кнопки — это пример того, что называется *событием*.

Событие (event) — это то, что происходит во время работы программы. Например, щелчок кнопкой мыши — это событие `Click`, двойной щелчок мышью — событие `DoubleClick`.

В табл. 2.9 приведены некоторые события, возникающие в результате действий пользователя.

Таблица 2.9. События

Событие	Описание
Click	Щелчок кнопкой мыши
DoubleClick	Двойной щелчок кнопкой мыши
MouseDown	Нажатие кнопки мыши
MouseUp	Отпускание нажатой кнопки мыши
MouseMove	Перемещение указателя мыши
KeyPress	Нажатие клавиши
KeyDown	Нажатие клавиши. События <code>KeyDown</code> и <code>KeyPress</code> — это чередующиеся, повторяющиеся события, которые происходят до тех пор, пока не будет отпущена удерживаемая клавиша (в этот момент происходит событие <code>KeyUp</code>)
KeyUp	Отпускание нажатой клавиши
TextChanged	Признак, указывающий, изменился ли текст, находящийся в поле редактирования (изменилось значение свойства <code>Text</code>)
Load	Загрузка формы. Функция обработки этого события обычно используется для инициализации переменных, выполнения подготовительных действий
Paint	Событие происходит при появлении окна на экране в начале работы программы, после появления части окна, которая, например, была закрыта другим окном
Enter	Получение фокуса элементом управления
Leave	Потеря фокуса элементом управления

Следует понимать, что одни и те же действия, но выполненные над разными объектами, вызывают разные события. Например, нажатие клавиши (событие `KeyPress`) в поле ввода/редактирования **Сумма** и нажатие клавиши (также событие `KeyPress`) в поле **Срок** — это два разных события.

Функция обработки события

Реакцией на событие должно быть какое-либо действие. В Visual C# реакция на событие реализуется как *функция обработки события*. Таким образом, для того чтобы программа в ответ на действия пользователя выполняла некоторую работу, программист должен написать функцию (метод) обработки соответствующего события.

Процесс создания функции обработки события рассмотрим на примере обработки события Click для кнопки **Расчет**.

Чтобы создать функцию обработки события, сначала надо выбрать компонент, для которого создается функция обработки события. Для этого в окне конструктора формы надо сделать щелчок левой кнопкой мыши на нужном компоненте. Затем в окне **Properties** щелчком на кнопке **Events** (рис. 2.19) нужно открыть вкладку **Events**.

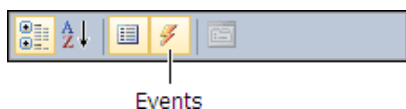


Рис. 2.19. Кнопка **Events**

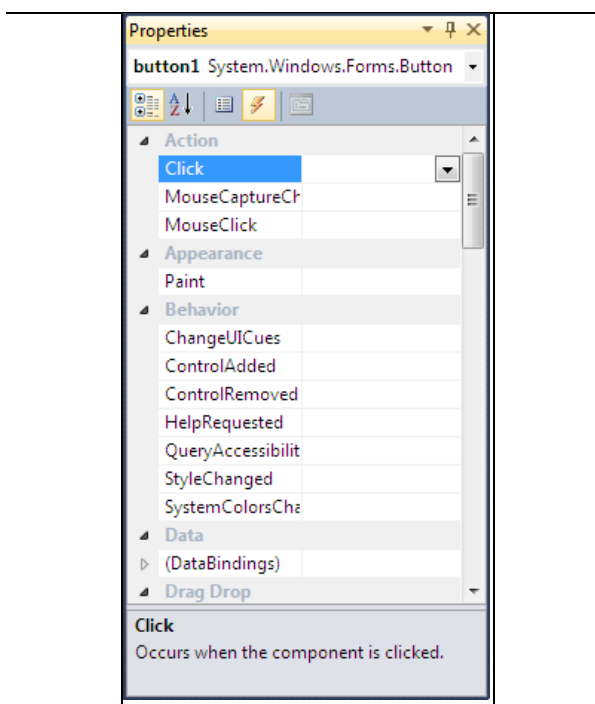


Рис. 2.20. На вкладке **Events** перечислены события, которые может воспринимать компонент

В левой колонке вкладки **Events** (рис. 2.20) перечислены события, которые может воспринимать выбранный компонент. Строго говоря, на вкладке **Events** указаны не события, а свойства, значением которых являются имена функций обработки соответствующих событий.

Для того чтобы создать функцию обработки события, нужно на вкладке **Events** выбрать событие (сделать щелчок мышью на имени события), в поле значения

свойства ввести имя функции обработки события (рис. 2.21) и нажать клавишу <Enter>.

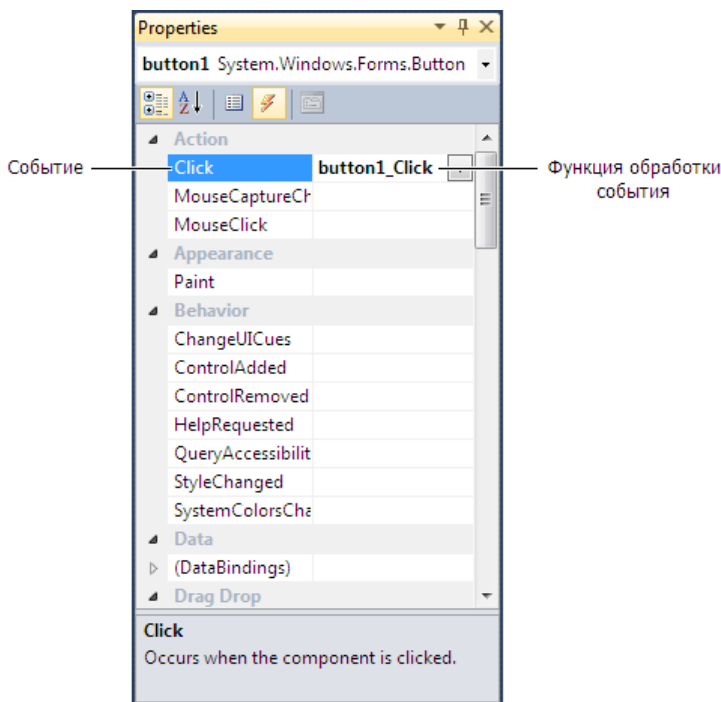


Рис. 2.21. Рядом с именем события надо ввести имя функции обработки события

В результате этих действий в модуль формы (cs-файл) будет добавлена функция (метод класса формы) обработки события и станет доступным окно редактора кода (рис. 2.22), в котором можно набирать инструкции, реализующие функцию обработки события.

Функция обработки события Click для кнопки **Расчет** (button1) приведена в листинге 2.1.

Листинг 2.1. Обработка события Click для кнопки *Расчет*

```
private void button1_Click(object sender, EventArgs e)
{
    double sum;           // сумма
    int    period;        // срок

    double percent;       // процентная ставка
    double profit;        // доход

    sum = System.Convert.ToDouble(textBox1.Text);
    period = System.Convert.ToInt32(textBox2.Text);
```

```

if (sum < 10000)
    percent = 8.5;
else
    percent = 12;

profit = sum * (percent/100/12) * period;

label3.Text = "Процентная ставка: " + percent.ToString("n") + "%\n" +
    "Доход: " + profit.ToString("c");
}

```

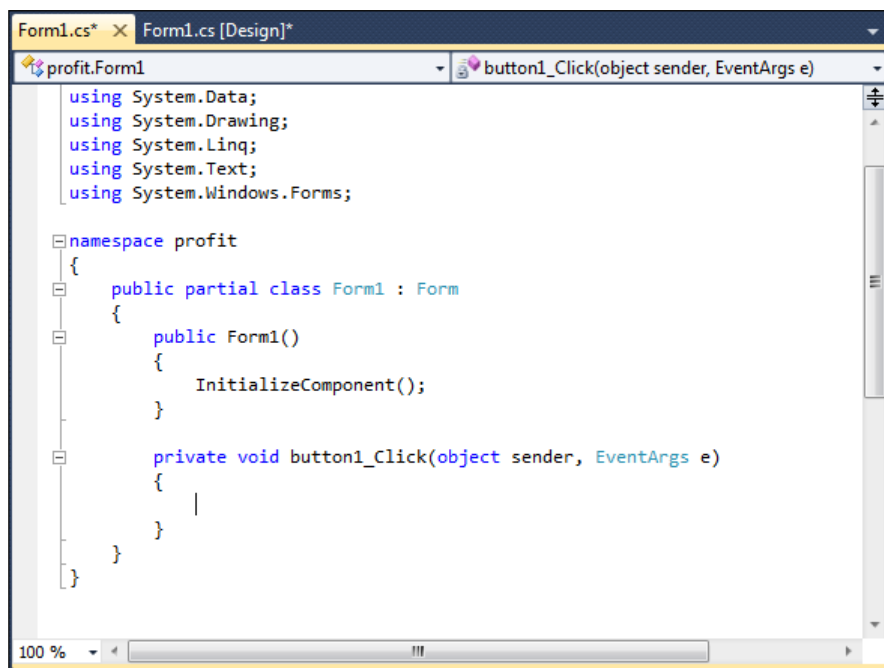


Рис. 2.22. Шаблон функции (метода) обработки события

Функция `button1_Click` вычисляет доход по вкладу и выводит результат расчета в поле компонента `label3`. Исходные данные (сумма и срок вклада) вводятся из полей редактирования `textBox1` и `textBox2` путем обращения к свойству `Text`. Значением свойства `Text` является строка, которая находится в поле редактирования. Свойство `Text` строкового типа, поэтому для преобразования строк в числа используются принадлежащие *пространству имен* `System.Convert` функции `ToDouble` и `ToInt32`. Следует обратить внимание, что функция `ToDouble` возвращает результат только в том случае, если строка, переданная ей в качестве параметра, является изображением дробного числа, что предполагает использование *запятой* в качестве десятичного разделителя (при стандартной для России настройке операционной системы). Аналогично, параметр функции `ToInt32` должен представлять собой строку, являющуюся изображением целого числа.

Другие функции преобразования строк приведены в табл. 2.10.

Таблица 2.10. Функции преобразования строк

Функция	Значение
ToSingle (s), ToDouble (s)	Дробное типа Single, Double
ToByte (s), ToInt16 (s), ToInt32 (s), ToInt64 (s),	Целое типа Byte, Int16, Int32, Int64
ToUInt16 (s), ToUInt32 (s), ToUInt64 (s)	Целое типа UInt16, UInt32, UInt64

ПРОСТРАНСТВО ИМЕН

Концепция пространства имен является развитием концепции модулей. Пространство имен позволяет избежать конфликта имен, дает программисту свободу в выборе идентификаторов. Так, например, при объявлении функции можно не заботиться об уникальности ее имени, достаточно объявить эту функцию в новом пространстве имен.

В приведенной в листинге 2.1 функции для преобразования строки в дробное число используется функция ToDouble. Она принадлежит пространству имен System.Convert, на что указывает префикс перед именем функции (строго говоря, функция ToDouble — это метод объекта Convert, который принадлежит пространству имен System).

Пространство имен (namespace) — это контейнер (модуль), который предоставляет программе, использующей этот модуль, свои объекты (типы, функции, константы и т. д.). Например, пространство имен System.Windows.Forms содержит объекты Label, TextBox, Button и др.

Каждый объект является элементом какого-либо пространства имен. Например, поле редактирования, объект типа TextBox, является элементом или, как принято говорить, принадлежит пространству имен System.Windows.Forms.

Пространства имен, которые использует программа, указывается в инструкции using. Например, в начале модуля формы (cs-файл) есть ссылки на пространства имен System, System.Windows.Forms, System.Drawing и др.

Для того чтобы получить доступ к объекту пространства имен (например, методу или константе), следует перед именем объекта указать идентификатор пространства имен, которому принадлежит объект, разделив идентификатор и имя объекта точкой.

Например, инструкция

```
n = System.Convert.ToSingle(TextBox1.Text);
```

показывает, что для преобразования строки в число используется метод ToSingle объекта Convert, который принадлежит пространству имен System.

Вычисленные значения процентной ставки и величины дохода выводятся в поле label3 путем присваивания значения свойству Text. Для преобразования дробного числа в строку (свойство Text строкового типа) используется функция (метод) ToString. Параметр метода ToString задает формат строки-результата: "c" — финансовый (от англ. currency); "n" — числовой (от англ. number). Следует обратить внимание, что при использовании финансового формата после числового значения

выводится обозначение денежной единицы (в соответствии с настройкой операционной системы). В табл. 2.11 приведены возможные форматы представления числовой информации.

Таблица 2.11. Форматы представления чисел

Параметр функции ToString	Формат	Пример
"c"	Currency — финансовый (денежный). Используется для представления денежных величин. Обозначение денежной единицы, разделитель групп разрядов, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28 р.
"e"	Scientific (exponential) — научный. Используется для представления очень маленьких или очень больших чисел. Разделитель целой и дробной частей числа задается в настройках операционной системы	5,50528+E004
"f"	Fixed — число с фиксированным десятичным разделителем. Используется для представления дробных чисел. Количество цифр дробной части, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28
"n"	Number — числовой. Используется для представления дробных чисел. Количество цифр дробной части, символ-разделитель групп разрядов, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28
"g"	General — универсальный формат. Похож на Number, но разряды не разделены на группы	55055,275
"r"	Roundtrip — без округления. В отличие от формата N, этот формат не выполняет округления (количество цифр дробной части зависит от значения числа)	55 055,2775

Структура проекта

Проект представляет собой совокупность файлов, которые компилятор использует для создания выполняемого файла. Структура проекта отображается в окне **Solution Explorer** (рис. 2.23).

Основными элементами проекта являются:

- ◆ главный модуль приложения (файл Program.css);
- ◆ модули форм.

Главный модуль

В главном модуле находится функция `Main`, с которой начинается выполнение программы. Функция `Main` создает стартовую форму (имя класса стартовой формы

указывается в качестве параметра метода `Run()`, в результате чего на экране появляется окно программы. Главный модуль программы "Доход" приведен в листинге 2.2.

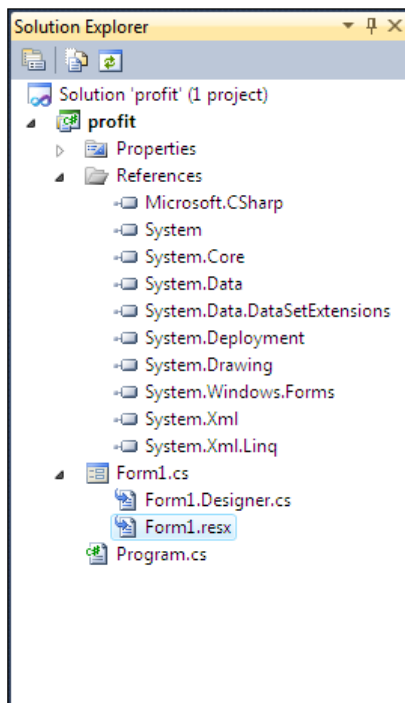


Рис. 2.23. Структура проекта отображается в окне **Solution Explorer**

Листинг 2.2. Главный модуль программы "Доход" (Program.cs)

```
using System.Windows.Forms;

namespace profit
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Модуль формы

Модуль формы содержит объявление класса формы. Физически модуль формы разделен на два файла: Form1.cs и Form1.Designer.cs (листинги 2.3 и 2.4 соответственно). В файле Form1.cs находятся функции (методы класса формы) обработки событий формы и ее компонентов. В файле Form1.Designer.cs (чтобы его увидеть, надо в окне **Solution Explorer** сделать двойной щелчок на имени файла) находится объявление класса формы, в том числе сформированная дизайнером формы функция `InitializeComponent`, обеспечивающая создание и настройку компонентов. Следует обратить внимание на секцию **Windows Form Designer generated code** (секция — фрагмент кода, находящийся между директивами `#region` и `#endregion`). В ней находится функция `InitializeComponent`, обеспечивающая непосредственно создание и инициализацию формы и компонентов.

Листинг 2.3. Модуль формы (файл Form1.cs)

```
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 profit
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            double sum;          // сумма
            int    period;       // срок

            double percent;      // процентная ставка
            double profit;       // доход

            sum = Convert.ToDouble(textBox1.Text);
            period = Convert.ToInt32(textBox2.Text);

            if (sum < 10000)
                percent = 8.5;
```

```

        else
            percent = 12;

        profit = sum * (percent/100/12) * period;

        label3.Text =
            "Процентная ставка: " + percent.ToString("n") + "%\n" +
            "Доход: " + profit.ToString("c");
    }
}
}

```

Листинг 2.4. Модуль формы (файл Form1.Designer.cs)

```

namespace profit
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
        /// disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.label1 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();

```

```

this.textBox1 = new System.Windows.Forms.TextBox();
this.textBox2 = new System.Windows.Forms.TextBox();
this.button1 = new System.Windows.Forms.Button();
this.label3 = new System.Windows.Forms.Label();
this.SuspendLayout();
//
// label1
//
this.label1.AutoSize = true;
this.label1.Location = new System.Drawing.Point(9, 18);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(88, 16);
this.label1.TabIndex = 0;
this.label1.Text = "Сумма (pyб.);";
//
// label2
//
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(21, 47);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(76, 16);
this.label2.TabIndex = 1;
this.label2.Text = "Срок (мес.):";
//
// textBox1
//
this.textBox1.Location = new System.Drawing.Point(103, 15);
this.textBox1.Name = "textBox1";
this.textBox1.Size = new System.Drawing.Size(100, 23);
this.textBox1.TabIndex = 2;
//
// textBox2
//
this.textBox2.Location = new System.Drawing.Point(103, 44);
this.textBox2.Name = "textBox2";
this.textBox2.Size = new System.Drawing.Size(100, 23);
this.textBox2.TabIndex = 3;
//
// button1
//
this.button1.Location = new System.Drawing.Point(24, 97);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(75, 23);
this.button1.TabIndex = 4;
this.button1.Text = "Пачет";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click +=
        new System.EventHandler(this.button1_Click);

```



```

//
// label3
//
this.label3.Location = new System.Drawing.Point(21, 135);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(228, 64);
this.label3.TabIndex = 5;
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(7F, 16F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(359, 204);
this.Controls.Add(this.label3);
this.Controls.Add(this.button1);
this.Controls.Add(this.textBox2);
this.Controls.Add(this.textBox1);
this.Controls.Add(this.label2);
this.Controls.Add(this.label1);
this.Font = new System.Drawing.Font
    ("Tahoma", 9.75F, System.Drawing.FontStyle.Regular,
    System.Drawing.GraphicsUnit.Point, ((byte) 204));
this.FormBorderStyle =
    System.Windows.Forms.FormBorderStyle.FixedSingle;
this.Margin = new System.Windows.Forms.Padding(3, 4, 3, 4);
this.MinimizeBox = false;
this.Name = "Form1";
this.StartPosition =
    System.Windows.Forms.FormStartPosition.CenterScreen;
this.Text = "Доход";
this.ResumeLayout(false);
this.PerformLayout();

}

#endregion

private System.Windows.Forms.Label label1;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.TextBox textBox1;
private System.Windows.Forms.TextBox textBox2;
private System.Windows.Forms.Button button1;
private System.Windows.Forms.Label label3;
}
}

```

Сохранение проекта

Как было сказано раньше, в момент создания проекта среда разработки в папке временных проектов (по умолчанию это `C:\Users\User\AppData\Local\Temporary Projects`, где *User* — имя пользователя в системе) создает каталог проекта. Чтобы программист мог работать с программой в дальнейшем, проект надо сохранить явно. Для этого в меню **File** надо выбрать команду **Save All** и в появившемся окне **Save Project** нажать кнопку **Save** (рис. 2.24).

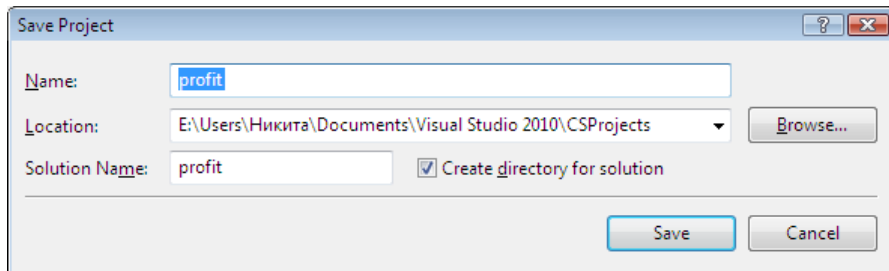


Рис. 2.24. Сохранение проекта

Следует обратить внимание, что в момент сохранения проекта в папке, имя которой указано в поле **Location**, для сохраняемого проекта будет создана новая папка (если установлен флажок **Create directory for solution**).

Компиляция

Процесс преобразования исходной программы в выполняемую называется *компиляцией* или построением (build). Укрупненно процесс построения программы можно представить как последовательность двух этапов: компиляция и компоновка. На этапе компиляции выполняется перевод исходной программы (модулей) в некоторое внутреннее представление. На этапе компоновки — объединение модулей в единую программу.

Процесс построения программы активизируется в результате выбора в меню **Debug** команды **Build solution**, а также в результате запуска программы из среды разработки (меню **Debug**, команда **Start Debugging**), если с момента последней компиляции в программу были внесены изменения.

Результат компиляции отражается в окне **Error List**. Если в программе нет ошибок, то по завершении процесса компиляции окно **Error List** выглядит так, как показано на рис. 2.25.

Если в процессе построения в программе обнаруживаются ошибки, то в окне **Error List** (рис. 2.26) выводится их список. Чтобы перейти к фрагменту кода, содержащего ошибку, надо сделать двойной щелчок левой кнопкой мыши в строке сообщения об этой ошибке.

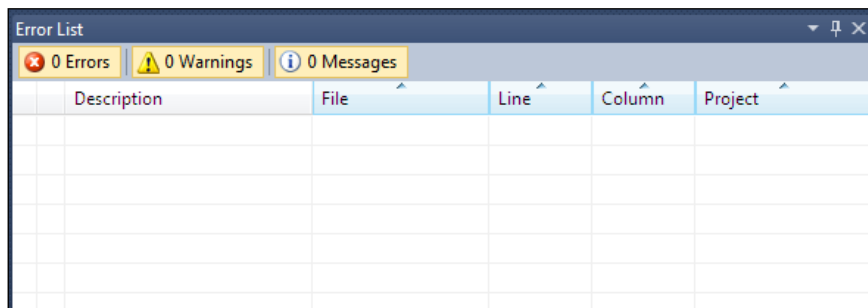


Рис. 2.25. Результат построения (в программе ошибок нет)

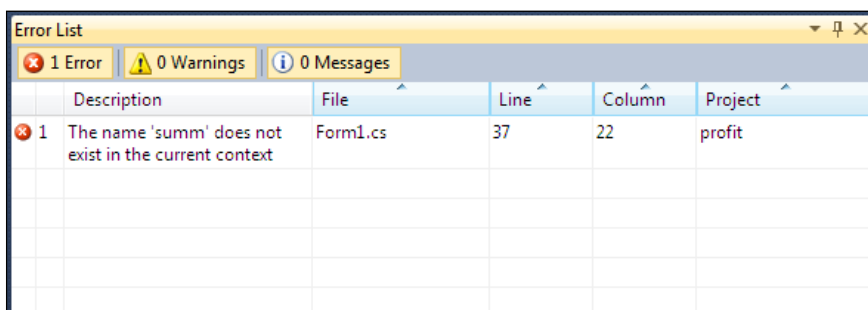


Рис. 2.26. Результат построения (в программе есть ошибка)

Ошибки

Компилятор генерирует выполняемую программу (exe-файл) только в том случае, если в исходной программе (в тексте) нет ошибок.

Если в программе есть ошибки, то программист должен их устранить. Процесс устранения ошибок носит итерационный характер. Обычно сначала устраняются наиболее очевидные ошибки, например, объявляются необъявленные переменные, затем, после выполнения повторной компиляции, — остальные.

В табл. 2.12 приведены сообщения компилятора о типичных ошибках.

Таблица 2.12. Сообщения компилятора об ошибках

Сообщение компилятора	Вероятная причина ошибки
The name <i>идентификатор</i> does not exist in the current context (В текущем контексте имя не существует)	1. Используемая в программе переменная не объявлена. 2. Ошибка при записи имени переменной. Например, объявлена переменная <i>sum</i> , а в тексте программы написано: <i>Sum</i>
Cannot implicitly convert type <i>type1</i> to <i>type2</i> (Невозможно преобразовать значение типа <i>type1</i> в значение типа <i>type2</i>) Пример: Cannot implicitly convert type 'double' to 'int'	В инструкции присваивания тип выражения не соответствует типу переменной, которой присваивается значение. Например, если переменные <i>n</i> и <i>m</i> целого типа, то инструкция <i>n = m/12</i> неверная, т. к. выражение <i>m/12</i> дробное

Таблица 2.12 (окончание)

Сообщение компилятора	Вероятная причина ошибки
Use of unassigned local variable (Используется локальная переменная, которой не присвоено начальное значение)	В программе нет инструкции, присваивающей переменной начальное значение
<i>Пространство имен</i> does not contain definition for <i>Идентификатор</i> (Пространство имен <i>Пространство имен</i> не содержит определение идентификатора <i>Идентификатор</i>)	Неправильно (например, не в том регистре) записано имя пространства имен или идентификатор (например, имя функции) ему принадлежащий. Пример: <code>System.Convert.ToInt32(textBox1.Text)</code> — неправильно записано имя функции. Должно быть <code>ToInt32</code>
';' expected (ожидается символ "точка с запятой")	После инструкции нет символа "точка с запятой"

Следует обратить внимание на то, что компилятор языка C# *различает* прописные и строчные буквы. Запись имени переменной или функции "не в том регистре" — типичная причина ошибок в программе.

Предупреждения

В программе могут быть не только ошибки, но и неточности. Например, инструкция присваивания целой переменной дробного значения формально является верной. Присвоить значение переменной можно, но что делать с дробной частью? Отбросить или округлить? Что хотел сделать программист, записав эту инструкцию?

При обнаружении в программе неточностей компилятор выводит предупреждения — Warnings. Например, при обнаружении не объявленной, но не используемой переменной выводится сообщение: `unreferenced local variable`. Действительно, зачем объявлять переменную и не использовать ее?

В табл. 2.13 приведены предупреждения и подсказки компилятора о типичных неточностях в программе.

Таблица 2.13. Предупреждения и подсказки компилятора

Сообщение	Причина
The variable ... is declared but never used	Переменная объявлена, но не используется
The variable ... is assigned but its value is never used	Переменной присвоено значение, но оно не используется

Запуск программы

Пробный запуск программы можно выполнить из Visual Studio, не завершая работу со средой разработки. Для этого в меню **Debug** надо выбрать команду **Start Debugging**. Можно также сделать щелчок на находящейся в панели инструментов **Debug** кнопке **Start Debugging** (рис. 2.27) или нажать клавишу <F5>.



Рис. 2.27. Чтобы запустить программу, сделайте щелчок на кнопке **Start Debugging**

Исключения

Ошибки, возникающие во время работы программы, называют *исключениями*. В большинстве случаев причиной исключений (exception) являются неверные данные. Например, если в поле **Сумма** окна программы "Доход" ввести, скажем, 100.50 и сделать щелчок на кнопке **Расчет**, то на экране появится окно (рис. 2.28) с сообщением о возникновении исключения `FormatException: "Input string was not in correct format"` ("Неверный формат введенной строки"). Кроме этого, среда разработки делает активным окно редактора кода, в котором выделяется инструкция программы, при выполнении которой произошла ошибка (возникло исключение).

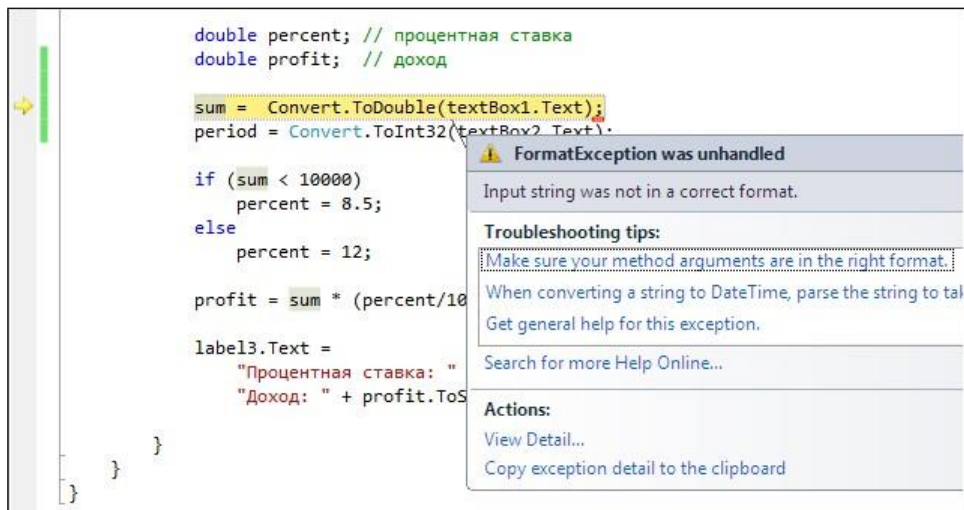


Рис. 2.28. Пример сообщения об исключении – ошибке, произошедшей во время работы программы (программа запущена из среды разработки)

Причина возникновения исключения в рассматриваемом примере в следующем. Преобразование строки в число выполняет функция `ToDouble`. Эта функция работа-

ет правильно, если ее параметром действительно является строковое представление дробного числа, что при стандартной для России настройке операционной системы предполагает использование в качестве десятичного разделителя *запятой*. В рассматриваемом примере строка 100.50 не является строковым представлением дробного числа, т. к. в качестве десятичного разделителя указана *точка*, и, поэтому, возникает исключение `FormatException` — ошибка формата. Исключение "ошибка формата" произойдет и в том случае, если в поле **Срок** будет введено дробное значение. Причина — попытка преобразовать в целое значение строку, которая не является изображением целого числа. Это же исключение произойдет и в том случае, если какое-либо из полей ввода оставить незаполненным.

Для того чтобы остановить программу, во время работы которой возникло исключение, надо в меню **Debug** выбрать команду **Stop Debugging** или нажать комбинацию клавиш `<Shift>+<F5>`.

Если программа запущена из операционной системы (из Windows), то при возникновении исключения так же, как и в случае запуска программы из среды разработки, выводится сообщение об ошибке (рис. 2.29). Чтобы остановить работу программы, надо нажать кнопку **Quit**. Щелчок на кнопке **Continue** разрешает продолжить выполнение программы, несмотря на возникшую ошибку.

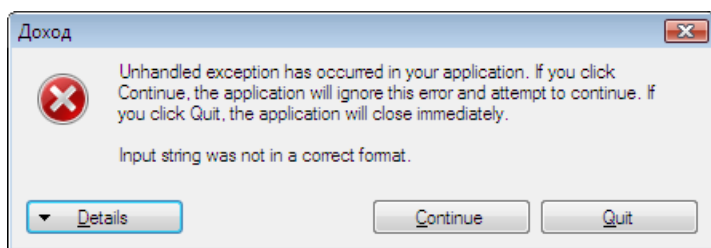


Рис. 2.29. Пример сообщения о возникновении исключения (программа запущена из операционной системы)

Обработка исключения

По умолчанию обработку исключений берет на себя автоматически добавляемый в выполняемую программу код, который обеспечивает вывод сообщения об ошибке и завершение работы программы, при выполнении которой возникло исключение. Вместе с тем программист может поместить в программу код, который выполнит обработку исключения.

В простейшем случае инструкция обработки исключения в общем виде выглядит так:

```
try
{
    // Здесь инструкции, при выполнении которых
    // может возникнуть исключение
}
```

```
catch (ТипИсключения e)
{
    // Здесь инструкции обработки исключения
}
```

Ключевое слово `try` указывает, что далее следуют инструкции, при выполнении которых возможно возникновение исключений, и что обработку этих исключений берет на себя программа. Слово `catch` обозначает начало секции обработки исключений. После слова `catch` указывается тип исключения, обработку которого берет на себя программа. Далее следуют инструкции, обеспечивающие обработку исключения. Нужно обратить внимание на то, что инструкции секции `try`, следующие за той, при выполнении которой возникло исключение, после обработки исключения не выполняются.

В табл. 2.14 перечислены некоторые из возможных исключений и указаны вероятные причины их возникновения.

Таблица 2.14. Типичные исключения

Исключение	Возникает
<code>FormatException</code> – ошибка формата (преобразования)	При выполнении преобразования, если преобразуемая величина не может быть приведена к требуемому типу. Наиболее часто возникает при преобразовании строки символов в число
<code>IndexOutOfRangeException</code> – выход значения индекса за допустимые границы	При обращении к несуществующему элементу массива
<code>ArgumentOutOfRangeException</code> – выход значения аргумента за допустимые границы	При обращении к несуществующему элементу данных, например, при выполнении операций со строками
<code>OverflowException</code> – переполнение	Если результат выполнения операции выходит за границы допустимого диапазона, а также при выполнении операции деления, если делитель равен нулю

В качестве примера обработки исключения в листинге 2.5 приведена функция обработки события `Click` для кнопки **Расчет** программы "Доход". При возникновении исключения `FormatException` программа определяет причину (какое из полей формы незаполнено или содержит неверные данные) и выводит соответствующее сообщение (рис. 2.30).

Листинг 2.5. Щелчок на кнопке *Расчет* (пример обработки исключения)

```
private void button1_Click(object sender, EventArgs e)
{
    double sum;        // сумма
    int    period;     // срок

    double percent;    // процентная ставка
    double profit;     // доход
```

```

try
{
    sum = Convert.ToDouble(textBox1.Text);
    period = Convert.ToInt32(textBox2.Text);
    if (sum < 10000)
        percent = 8.5;
    else
        percent = 12;

    profit = sum * (percent / 100 / 12) * period;

    label3.Text =
        "Процентная ставка: " + percent.ToString("n") + "%\n" +
        "Доход: " + profit.ToString("c");
}
catch (FormatException ex)
{
    // MessageBox.Show(ex.Message);
    if ((textBox1.Text.Length == 0) || (textBox2.Text.Length == 0))
        MessageBox.Show("Оба поля должны быть заполнены.",
            "Доход", MessageBoxButtons.OK,
            MessageBoxIcon.Error);

    else
        MessageBox.Show("Ошибка в исходных данных. " +
            "В поле Сумма надо ввести целое или дробное число " +
            "(в качестве десятичного разделителя используйте " +
            "запятую), в поле Срок - целое.",
            "Доход",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
};
}

```

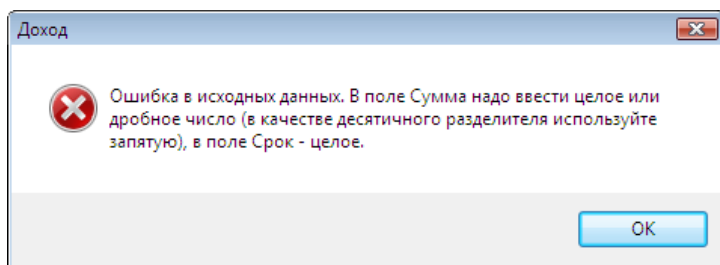


Рис. 2.30. Пример сообщения об ошибке

В приведенной функции обработки события Click для вывода сообщения о неверных данных используется функция `MessageBox.Show`, инструкция вызова которой в общем виде выглядит так:

```

r = MessageBox.Show(Сообщение, Заголовок, Кнопки, ТипСообщения,
    КнопкаПоУмолчанию)

```


где:

- ◆ *Сообщение* — текст сообщения;
- ◆ *Заголовок* — текст в заголовке окна сообщения;
- ◆ *Кнопки* — кнопки, отображаемые в окне сообщения (табл. 2.15);
- ◆ *Тип* — тип сообщения. Сообщение может быть информационным, предупреждающим или сообщением об ошибке. Каждому типу сообщения соответствует значок (табл. 2.16);
- ◆ *КнопкаПоУмолчанию* — порядковый номер кнопки, на которой находится фокус при появлении окна сообщения на экране (табл. 2.17).

Значение (тип `System.Windows.Forms.DialogResult`), возвращаемое функцией `MessageBox.Show`, позволяет определить, какая кнопка была нажата пользователем для завершения диалога (табл. 2.18). Если в окне сообщения отображается одна кнопка (очевидно, что в этом случае не нужно проверять, какую кнопку нажал пользователь), то функцию `MessageBox.Show` можно вызвать как процедуру.

Таблица 2.15. Идентификаторы кнопок

Значение параметра	Кнопки, отображаемые в окне сообщения
<code>MessageBoxButtons.OK</code>	OK
<code>MessageBoxButtons.YesNo</code>	Yes, No (Да, Нет)
<code>MessageBoxButtons.YesNoCancel</code>	Yes, No, Cancel (Да, Нет, Отменить)

Таблица 2.16. Тип сообщения




Сообщение	Тип сообщения	Значок
Warning – Внимание	<code>MessageBoxIcon.Warning</code>	
Error – Ошибка	<code>MessageBoxIcon.Error</code>	
Information – Информация	<code>MessageBoxIcon.Information</code>	

Таблица 2.17. Активная кнопка

Значение параметра	Номер активной кнопки
<code>System.Windows.Forms.MessageBoxDefaultButton.Button1</code>	1
<code>System.Windows.Forms.MessageBoxDefaultButton.Button2</code>	2
<code>System.Windows.Forms.MessageBoxDefaultButton.Button3</code>	3

Таблица 2.18. Значения функции *MessageBox.Show*

Значение	Нажатая кнопка
System.Windows.Forms.DialogResult.Yes	Yes
System.Windows.Forms.DialogResult.No	No
System.Windows.Forms.DialogResult.Cancel	Cancel

Внесение изменений

Программу "Доход" можно усовершенствовать. Например, сделать так, чтобы в поля редактирования пользователь мог ввести только числа (в поле **Сумма** — дробное число, в поле **Срок** — целое), чтобы в результате нажатия клавиши <Enter> в поле **Сумма** курсор переходил в поле **Срок**, а при нажатии этой же клавиши в поле **Срок** становилась активной кнопка **Расчет**. Кроме этого, можно сделать так, чтобы кнопка **Расчет** становилась доступной только после ввода данных в оба поля редактирования.

Чтобы внести изменения в программу, нужно открыть соответствующий проект. Для этого надо в меню **File** выбрать команду **Open Project**, открыть папку проекта и сделать щелчок на значке файла проекта (рис. 2.31).

Нужный проект для загрузки можно выбрать также из списка проектов, над которыми в последнее время работал программист. Этот список становится доступным в результате выбора в меню **File** команды **Recent Projects and Solutions**.

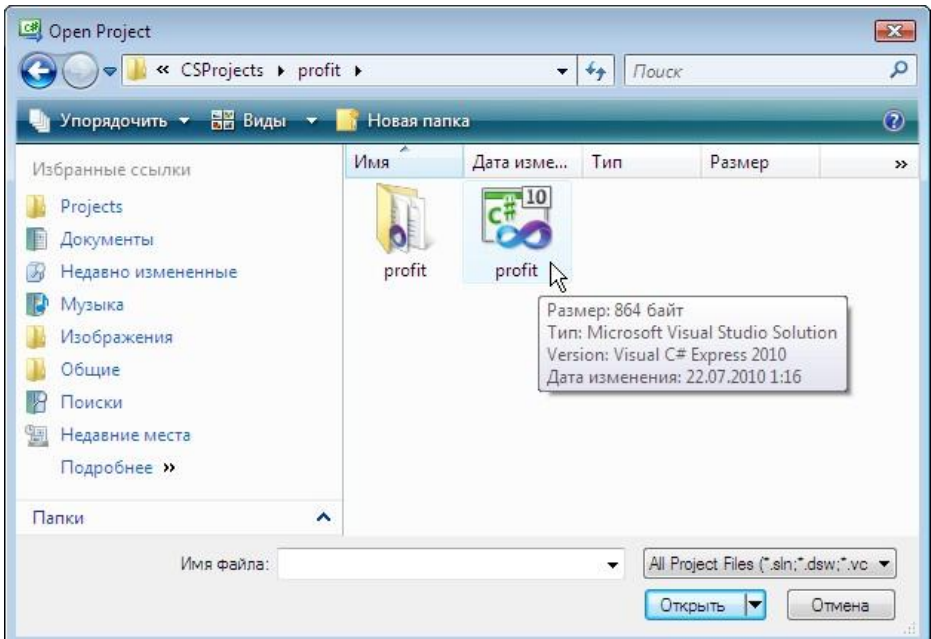


Рис. 2.31. Загрузка проекта

Чтобы программа "Доход" работала так, как было описано ранее, надо создать функции обработки событий `KeyPress` и `TextChanged` для полей редактирования (компонентов `textBox1` и `textBox2`). Функция обработки события `KeyPress` (для каждого компонента своя) фильтрует символы, вводимые пользователем. Она проверяет символ нажатой клавиши (символ передается в процедуру обработки события через параметр `e`) и, если символ "запрещен", присваивает значение `True` свойству `Handled`. В результате "запрещенный" символ в поле редактирования не появляется. Процедура обработки события `TextChanged` (событие возникает, если текст, находящийся в поле редактирования, изменился, например, в результате нажатия какой-либо клавиши в поле редактирования) управляет доступностью кнопки **Расчет**. Она проверяет, есть ли данные в полях редактирования, и, если в каком-либо из полей данных нет, присваивает свойству `Enabled` кнопки `button1` значение `False` (тем самым делает кнопку недоступной). Следует обратить внимание, что действие, которое надо выполнить, если изменилось содержимое поля `textBox1`, ничем не отличается от действия, которое надо выполнить, если изменилось содержимое поля `textBox2`. Поэтому обработку события

`TextChanged` для обоих компонентов может выполнить *одна* функция. Чтобы *одна* функция могла обрабатывать события *разных* компонентов, сначала надо создать функцию обработки события для одного компонента, а затем указать эту функцию в качестве обработчика соответствующего события другого компонента (рис. 2.32).

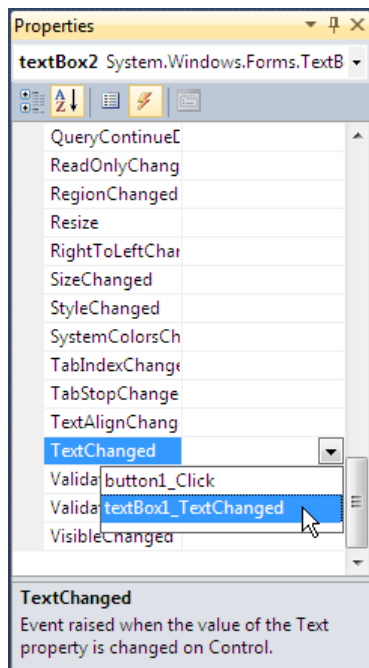


Рис. 2.32. Выбор функции обработки события: обработку события `TextChanged` компонента `textBox2` выполняет функция обработки этого же события компонента `textBox1`

Функции обработки указанных событий приведены в листинге 2.6. Обратите внимание, теперь в функции обработки события `Click` кнопки **Расчет** нет инструкций обработки исключений. Теперь они не нужны. Исключения не могут возникнуть, потому что пользователь не сможет ввести в поля редактирования неверные данные, а кнопка **Расчет** становится доступной только после того, как будут заполнены оба поля.

Листинг 2.6. Функции обработки событий

// щелчок на кнопке Расчет

```
private void button1_Click(object sender, EventArgs e)
```

```
{
```

```
    double sum;           // сумма
```

```
    int    period;       // срок
```

```
    double percent; // процентная ставка
```

```
    double profit;  // доход
```

```
    sum = Convert.ToDouble(textBox1.Text);
```

```
    period = Convert.ToInt32(textBox2.Text);
```

```
    if (sum < 10000)
```

```
        percent = 8.5;
```

```
    else
```

```
        percent = 12;
```

```
    profit = sum * (percent / 100 / 12) * period;
```

```
    label3.Text = "Процентная ставка: " + percent.ToString("n") + "%\n" +
```

```
        "Доход: " + profit.ToString("c");
```

```
}
```

// нажатие клавиши в поле Сумма

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
```

```
{
```

```
    if ((e.KeyChar >= '0') && (e.KeyChar <= '9')) // цифра  
        return;
```

```
    // "Правильный" десятичный разделитель — запятая.
```

```
    // Заменяем точку запятой
```

```
    if (e.KeyChar == '.') e.KeyChar = ',';
```

```
    if (e.KeyChar == ',')
```

```
    {
```

```
        // Нажата запятая. Проверим,
```

```
        // может, запятая уже есть в поле редактирования
```

```
        if ((textBox1.Text.IndexOf(',') != -1) ||
```

```
            (textBox1.Text.Length == 0))
```

```
        {
```

```
            // Запятая уже есть.
```

```
            // Запретить ввод еще одной
```

```
            e.Handled = true;
```

```
        }
```

```
    return;
```

```
}
```

```

if (Char.IsControl(e.KeyChar))
{
    if (e.KeyChar == (char)Keys.Enter)
    {
        // Нажата клавиша <Enter>.
        // Переместить курсор в поле Срок
        textBox2.Focus();
    }
    return;
}

// остальные символы запрещены
e.Handled = true;
}

// нажатие клавиши в поле Срок
private void textBox2_KeyPress(object sender, KeyPressEventArgs e)
{
    // в поле Срок можно ввести только целое число
    if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))
        // цифра
        return;

    if (Char.IsControl(e.KeyChar))
    {
        if (e.KeyChar == (char)Keys.Enter)
        {
            // Нажата клавиша <Enter>.
            // Переместить фокус на кнопку Расчет
            button1.Focus();
        }
        return;
    }

    // остальные символы запрещены
    e.Handled = true;
}

// Эта функция обрабатывает событие TextChanged (изменился текст в поле
// редактирования) обоих компонентов TextBox.
// Сначала надо обычным образом создать функцию обработки события
// TextChanged для компонента textBox1, а затем указать ее в качестве
// обработчика события TextChanged для компонента textBox2
private void textBox1_TextChanged(object sender, EventArgs e)
{
    label3.Text = ""; // очистить поле отображения
    // результата расчета

```

```

if ((textBox1.Text.Length == 0) || (textBox2.Text.Length == 0))
    // В поле редактирования нет данных.
    // Сделать кнопку Расчет недоступной
    button1.Enabled = false;
else
    // Сделать кнопку Расчет доступной
    button1.Enabled = true;
}

```

В листинге 2.7 приведен конструктор формы. Он показывает, как можно выполнить настройку компонентов формы "в коде".

Листинг 2.7. Конструктор класса формы программы "Доход"

```

public Form1 ()
{
    InitializeComponent();

    // сделать кнопку Расчет недоступной
    button1.Enabled = false;
}

```

Завершение работы над проектом

После того как приложение будет отлажено, можно выполнить его сборку. Для этого в меню **Debug** надо выбрать команду **Build**. В результате описанных действий в подкаталоге bin\Release каталога проекта будет создан выполняемый (exe) файл программы.

Для завершения работы над проектом надо в меню **File** выбрать команду **Close Solution**.

Установка приложения на другой компьютер

В простейшем случае приложение Windows Forms представляет собой единственный exe-файл (в .NET-терминологии — сборку). Таким образом, чтобы установить созданное в Microsoft Visual C# приложение на другой компьютер, достаточно перенести (скопировать) на диск этого компьютера exe-файл. Вместе с тем, необходимо помнить: для того чтобы .NET-приложение могло работать на другом компьютере, на нем должна быть установлена платформа .NET Framework соответствующей версии (по умолчанию проекты Visual Studio 2010 компилируются в режиме использования Microsoft .NET Framework 4.0). Если на компьютере пользователя платформа .NET Framework не установлена, то при запуске приложения будет выведено сообщение о необходимости установить .NET Framework (рис. 2.33).



Рис. 2.33. Сообщение об ошибке при попытке запустить приложение на компьютере, на котором нет Microsoft .NET Framework

Профессиональный подход к разработке программного обеспечения предполагает, что программист создает не только приложение, но и установщик — программу, обеспечивающую установку приложения на компьютер пользователя. О том, как создать установщик, рассказывается в *главе 10*.