Глава 5. Формы: проект FORMS

5.1. Настройка визуальных свойств форм. Открытие форм в обычном и модальном режиме

После создания проекта FORMS добавьте к нему две новые формы (они автоматически получат имена `Form2` и `Form3`). Разместите в форме `Form1` две кнопки `button1` и `button2`. Настройте свойства всех форм и компонентов (листинг 5.1, рис. 5.1–5.3). В начало описания класса `Form1` добавьте описания двух полей:

```
private Form2 form2 = new Form2();
private Form3 form3 = new Form3();
```

В конструктор класса `Form1` добавьте два оператора (листинг 5.2) и определите обработчики события `Shown` для формы `Form1` и события `Click` для кнопок `button1` и `button2` (листинг 5.3).

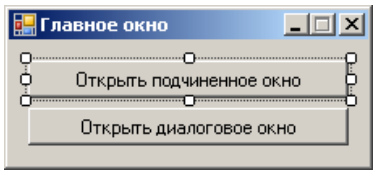

Рис. 5.1. Окончательный вид формы Form1 для проекта FORMS

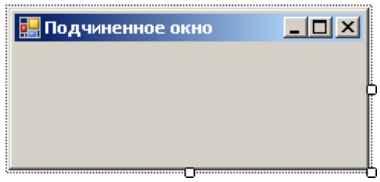

Рис. 5.2. Вид формы Form2 для проекта FORMS на начальном этапе разработки

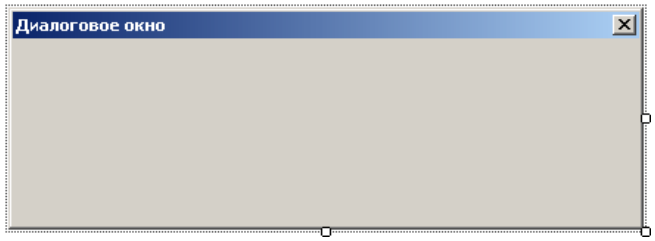

Рис. 5.3. Вид формы Form3 для проекта FORMS на начальном этапе разработки

Листинг 5.1. Настройка свойств

```
Form1: Text = Главное окно, MaximizeBox = False,
  FormBorderStyle = FixedSingle
Form2: Text = Подчиненное окно,
  StartPosition = Manual, ShowInTaskbar = False
Form3: Text = Диалоговое окно, MaximizeBox = False,
  MinimizeBox = False,
  FormBorderStyle = FixedDialog,
  StartPosition = CenterScreen,
  ShowInTaskbar = False
button1: Text = Открыть подчиненное окно
button2: Text = Открыть диалоговое окно
```

Листинг 5.2. Новый вариант конструктора формы `Form1`

```
public Form1()
{
  InitializeComponent();
  AddOwnedForm(form2);
  AddOwnedForm(form3);
}
```

Листинг 5.3. Обработчики `Form1.Shown`, `button1.Click` и `button2.Click`

```
private void Form1_Shown(object sender, EventArgs e)
{
  form2.Location = new Point(Right - 10, Bottom - 10);
}
private void button1_Click(object sender, EventArgs e)
{
  form2.Show();
}
```

```
private void button2_Click(object sender, EventArgs e)
{
  form3.ShowDialog();
}
```

Результат: программа содержит три формы, демонстрирующие основные типы окон в графических Windows-приложениях: окно фиксированного размера (класс `Form1`), окно переменного размера (класс `Form2`), диалоговое окно (класс `Form3`). На форме `Form1` размещены две кнопки (см. рис. 5.1); формы `Form2` и `Form3` пока не содержат компонентов (компоненты, изображенные на рис. 5.2 и 5.3, будут добавлены в формы на последующих этапах разработки проекта FORMS). Форма `Form1` является главной; она автоматически создается при запуске приложения и сразу отображается на экране. Кроме того, она создает две формы с именами `form2` и `form3` — экземпляры классов `Form2` и `Form3` соответственно.

Форма `form2` (подчиненная форма) вызывается из главной формы нажатием кнопки **Открыть подчиненное окно**; при этом она отображается в обычном режиме. Форма `form3` также является подчиненной; она вызывается нажатием кнопки **Открыть диалоговое окно** и отображается в модальном (диалоговом) режиме. Особенность модального режима состоит в том, что если некоторая форма приложения находится в этом режиме, то до ее закрытия нельзя переключаться на другие формы приложения (хотя возможно переключение на другие запущенные приложения). Для завершения программы надо закрыть ее главную форму.

Главная форма `Form1` имеет фиксированные размеры. Размеры подчиненной формы `form2` можно изменять; кроме того, форму `form2` можно разворачивать на весь экран. Визуальные свойства формы `form3` соответствуют стандартным свойствам диалогового окна: размеры формы `form3` нельзя изменять и, кроме того, в ее заголовке отображается только текст и кнопка закрытия (см. рис. 5.3).

При открытии формы `Form1` место для ее размещения выбирается операционной системой, форма `form2` отображается около правого нижнего угла формы `Form1` с небольшим наложением; форма `form3` всегда отображается в центре экрана.

Ошибка: после закрытия формы `form2` попытка ее повторного открытия приводит к исключительной ситуации с диагностикой "Cannot access a disposed object" ("Нет доступа к разрушенному объекту"). Это связано с тем, что закрытие формы, открытой в обычном режиме, приводит по умолчанию к ее разрушению. Заметим, что если форма открыта в диалоговом режиме, то ее разрушения при закрытии не происходит, в чем можно убедиться, несколько раз открыв и закрыв форму `form3`.

Исправление: определите обработчик события `FormClosing` для формы `Form2` (листинг 5.4).

Листинг 5.4. Обработчик `Form2.FormClosing`

```
private void Form2_FormClosing(object sender, FormClosingEventArgs e)
{
  if (e.CloseReason == CloseReason.UserClosing)
  {
    e.Cancel = true;
    Hide();
  }
}
```

Результат: теперь форму `form2`, как и форму `form3`, можно многократно закрывать и открывать в ходе выполнения программы.

5.2. Контроль за состоянием подчиненной формы. Воздействие подчиненной формы на главную

Измените метод `button1_Click` (листинг 5.5) и определите обработчик события `VisibleChanged` для формы `Form2` (листинг 5.6).

Листинг 5.5. Новый вариант метода `button1_Click`

```
private void button1_Click(object sender, EventArgs e)
{
  form2.Visible = !form2.Visible;
}
```

Листинг 5.6. Обработчик `Form2.VisibleChanged`

```
private void Form2_VisibleChanged(object sender, EventArgs e)
{
  Owner.Controls["button1"].Text = Visible ?
    "Закрыть подчиненное окно" : "Открыть подчиненное окно" ;
}
```

Результат: теперь текст кнопки `button1` и действия при ее нажатии зависят от того, отображается на экране подчиненное окно `form2` или нет: если подчиненное окно присутствует на экране, то оно исчезает, а если его на экране нет, то оно появляется. Подчиненное окно можно закрыть не только с помощью кнопки `button1`, но и любым стандартным способом, принятым в Windows (например, с помощью комбинации клавиш <Alt>+<F4>); при любом способе закрытия подчиненного окна заголовок кнопки `button1` будет изменен.

5.3. Компоненты, подстраивающиеся под размер окна

Разместите в форме `Form2` метку `label1` и настройте ее свойства (листинг 5.7; при установке свойств `Dock` и `TextAlign` на экране появляются графические окна выбора, в каждом из которых надо щелкнуть на центральном прямоугольном элементе). В результате форма Form2 примет вид, приведенный на рис. 5.4.

В начало описания класса `Form2` добавьте описание поля `count`:

```
private int count;
```

В метод `Form2_VisibleChanged` добавьте новые операторы (листинг 5.8).

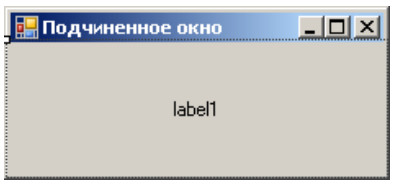

Рис. 5.4. Окончательный вид формы Form2 для проекта FORMS

Листинг 5.7. Настройка свойств

```
label1: AutoSize = False, Dock = Fill,
  TextAlign = MiddleCenter
```

Листинг 5.8. Добавление к методу `Form2_VisibleChanged`

```
if (Visible)
  label1.Text = "Окно открыто в " + (++count) + "-й раз.";
```

Результат: при изменении размеров подчиненного окна `Form2` размеры находящейся на нем метки `label1` изменяются так, чтобы она занимала всю внутреннюю (*клиентскую*) часть окна. Текст метки содержит информацию о том, сколько раз было открыто подчиненное окно.

5.4. Модальные и обычные кнопки диалогового окна

Разместите в форме `Form3` две метки (`label1` и `label2`), два поля ввода (`textBox1` и `textBox2`) и две кнопки (`button1` и `button2`). Настройте свойства добавленных компонентов, а также свойства формы `Form3` (листинг 5.9). При настройке взаимного расположения компонентов на форме (см. рис. 5.5) следует начать с полей ввода `textBox`, после чего выровнять по этим полям связанные с ними метки.

Определите обработчик события `Click` для кнопки `button2`, размещенной в форме `Form3` (листинг 5.10), а также обработчик события `Click` для кнопки `button2`, размещенной в форме `Form1` (листинг 5.11).

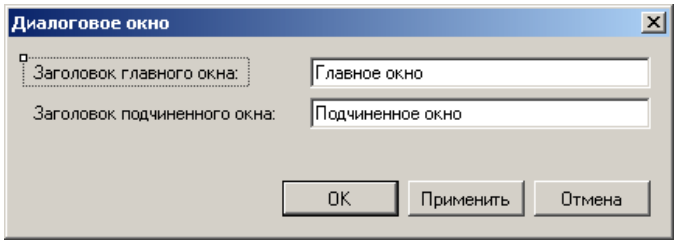

Рис. 5.5. Окончательный вид формы Form3 для проекта FORMS

Листинг 5.9. Настройка свойств

```
Form3: AcceptButton = button1,
  CancelButton = button3
label1: Text = Заголовок главного окна:
label2: Text = Заголовок подчиненного окна:
textBox1: Text = Главное окно
textBox2: Text = Подчиненное окно
button1: Text = OK, DialogResult = OK
button2: Text = Применить
button3: Text = Отмена
```

Листинг 5.10. Обработчик `button2.Click` (для кнопки формы `Form3`)

```
internal void button2_Click(object sender, EventArgs e)
// модификатор доступа изменен на internal
{
  Owner.Text = textBox1.Text;
  Owner.OwnedForms[0].Text = textBox2.Text;
}
```

Листинг 5.11. Обработчик `button2.Click` (для кнопки формы `Form1`)

```
private void button2_Click(object sender, EventArgs e)
{
  if (form3.ShowDialog() == DialogResult.OK)
  // - проверка, по нажатию какой кнопки
  // было закрыто диалоговое окно
    form3.button2_Click(this, EventArgs.Empty);
}
```

Результат: диалоговое окно `Form3` позволяет изменить заголовки главного и подчиненного окна. Заголовки окон изменяются либо при нажатии обычной кнопки **Применить**, либо при нажатии модальной кнопки **OK** (в последнем случае диалоговое окно закрывается). Окно также закрывается при нажатии модальной кнопки **Отмена**; в этом случае заголовки окон не изменяются. Вместо кнопки **OK** можно нажать клавишу <Enter>, вместо кнопки **Отмена** — <Esc>.

5.5. Установка активного компонента формы

Определите обработчик события `VisibleChanged` для формы `Form3` (листинг 5.12).

Листинг 5.12. Обработчик `Form3.VisibleChanged`

```
private void Form3_VisibleChanged(object sender, EventArgs e)
{
  if (Visible)
    ActiveControl = textBox1;
}
```

Результат: независимо от того, какой компонент диалогового окна был активным в момент его закрытия, при следующем открытии окна всегда оказывается активным поле ввода `textBox1`. Таким образом, диалоговое окно всегда отображается в одном и том же начальном состоянии. Подобное поведение желательно обеспечивать для любых диалоговых окон.

5.6. Запрос на подтверждение закрытия формы

Измените метод `Form2_FormClosing` (листинг 5.13).

Листинг 5.13. Новый вариант метода `Form2_FormClosing`

```
private void Form2_FormClosing(object sender, FormClosingEventArgs e)
{
  if (e.CloseReason == CloseReason.UserClosing)
  {
    e.Cancel = true;
    if (MessageBox.Show("Закрыть подчиненную форму?", "Подтверждение",
      MessageBoxButtons.YesNo, MessageBoxIcon.Question,
      MessageBoxDefaultButton.Button2) == DialogResult.Yes)
      Hide();
  }
}
```

Результат: перед закрытием подчиненного окна `form2` одним из способов, предусмотренных в системе Windows, в стандартном диалоговом окне **Подтверждение** выводится запрос на подтверждение закрытия (рис. 5.6). При выборе варианта **Нет** (который предлагается по умолчанию) закрытие подчиненного окна отменяется. При закрытии главного окна открытое подчиненное окно закрывается без запроса.

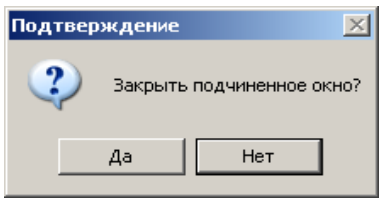

Рис. 5.6. Диалоговое окно **Подтверждение**

Недочет 1: при выборе в диалоговом окне варианта **Да** подчиненное окно закрывается, но главное окно не становится активным.

Это объясняется тем обстоятельством, что владельцем диалогового окна `MessageBox` является та форма, которая в данный момент была активной на экране (в нашем случае активной формой является `form2`), и именно эта форма должна активизироваться при закрытии окна `MessageBox`. Однако при выборе варианта **Да** форма `form2` закрывается, и поэтому ее активизация оказывается невозможной. В подобной ситуации ни одно окно на экране не будет активным, а главное окно нашей программы, скорее всего, будет заслонено окном среды Visual C#.

Исправление: в методе `Form2_FormClosing` замените оператор `Hide()` на следующий составной оператор:

```
{
  Hide();
```

```
  Owner.Activate();
}
```

Недочет 2: при закрытии подчиненного окна с помощью нажатия на кнопку `button1` главного окна запрос на подтверждение закрытия не выводится.

Это происходит потому, что при выполнении обработчика `button1_Click` не вызывается метод `Close` подчиненной формы (форма просто изменяет свой режим видимости), а следовательно, не выполняется и обработчик, связанный с методом `Close` подчиненной формы.

Исправление: измените метод `button1_Click` класса `Form1` (листинг 5.14).

Листинг 5.14. Новый вариант метода `button1_Click` класса `Form1`

```
private void button1_Click(object sender, EventArgs e)
{
  if (form2.Visible)
    form2.Close();
  else
    form2.Show();
}
```

Глава 6. Совместное использование обработчиков событий и работа с клавиатурой: проект CALC

6.1. Обработчик событий для нескольких компонентов

После создания проекта CALC разместите в форме `Form1` два поля ввода `TextBox`, две метки `Label` и пять кнопок `Button`. Для того чтобы уменьшить размер первых четырех кнопок одинаковым образом (см. рис. 6.1), следует после размещения этих кнопок на форме выделить их (например, охватив пунктирной рамкой) и изменить размер одной из них; при этом размер остальных выделенных кнопок изменится автоматически.

Настройте свойства формы и всех добавленных компонентов (листинг 6.1).

Определите обработчик события `Click` для кнопки `button1` (листинг 6.2) и свяжите полученный обработчик `button1_Click` с событием `Click` кнопок `button2`, `button3` и `button4`. Для этого надо выбрать в окне свойств **Properties** режим **Events**, для каждой из этих кнопок перейти на строку с событием **Click** и выбрать имя обработчика **button1_Click** из выпадающего списка (выполнять двойной щелчок на поле ввода *не следует*). Можно выполнить связывание сразу для всех трех кнопок; для этого надо предварительно выделить все эти кнопки на форме.

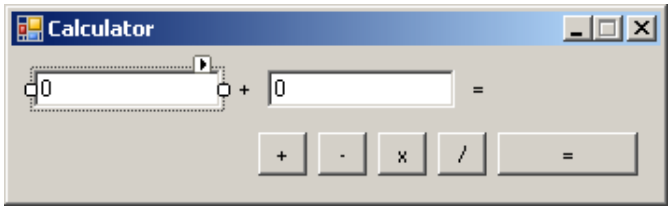

Рис. 6.1. Вид формы Form1 для проекта CALC на начальном этапе разработки

Листинг 6.1. Настройка свойств

```
Form1: Text = Калькулятор, MaximizeBox = False,
  FormBorderStyle = FixedSingle,
  StartPosition = CenterScreen
textBox1: Text = 0
textBox2: Text = 0
label1: Text = +
label2: Text = =
button1: Text = +
button2: Text = -
button3: Text = x
button4: Text = /
button5: Text = =
```

Листинг 6.2. Обработчик `button1.Click`

```
private void button1_Click(object sender, EventArgs e)
{
  label1.Text = (sender as Button).Text;
}
```

Результат: нажатие на любую кнопку с обозначением операции (+, –, x, /) приводит к отображению этой операции в метке `label1` между полями ввода `textBox1` и `textBox2`.

6.2. Вычисления с контролем правильности исходных данных

Определите обработчик события `Click` для кнопки `button5` (листинг 6.3).

Листинг 6.3. Обработчик `button5.Click`

```
private void button5_Click(object sender, EventArgs e)
{
  double x = 0,
    x1 = double.Parse(textBox1.Text),
    x2 = double.Parse(textBox2.Text);
  switch (label1.Text[0])
  {
    case '+':
      x = x1 + x2; break;
    case '-':
      x = x1 - x2; break;
    case 'x':
      x = x1 * x2; break;
    case '/':
      x = x1 / x2; break;
  }
  label2.Text = "= " + x;
}
```

Результат: при нажатии кнопки = указанное выражение вычисляется и отображается на экране (в метке `label2`). В качестве второго операнда при любой операции можно указывать число 0; при делении на 0 результат имеет вид **–бесконечность** или **бесконечность** (в зависимости от знака первого ненулевого операнда). Если оба операнда равны 0, то результат имеет значение **NaN** ("не число").

Недочет: если одно из полей ввода не содержит текст или этот текст нельзя преобразовать в число (например, **abc**), то при нажатии на кнопку = возникает *исключительная ситуация* (*исключение*). Если программа запущена в режиме с включенным отладчиком (режим **Debug**, запуск клавишей <F5>), то ее выполнение будет прервано, а в редакторе среды Visual C# будет подсвечен оператор, вызвавший ошибку. Возможные действия в этой ситуации подробно описаны в разд. 3.1.

Исправление: измените метод `button5_Click` (листинг 6.4).

Листинг 6.4. Новый вариант метода `button5_Click`

```
private void button5_Click(object sender, EventArgs e)
{
  double x = 0, x1, x2;
  if (!double.TryParse(textBox1.Text, out x1) ||
    !double.TryParse(textBox2.Text, out x2))
  {
    label2.Text = "= ERROR";
    return;
  }
  switch (label1.Text[0])
  {
    case '+':
      x = x1 + x2; break;
    case '-':
      x = x1 - x2; break;
    case 'x':
      x = x1 * x2; break;
    case '/':
      x = x1 / x2; break;
  }
  label2.Text = "= " + x;
}
```

Результат: теперь при попытке вычислить выражение с недопустимыми операндами в метке `label2` выводится текст **ERROR**; при этом прерывание выполнения программы не происходит, и окно с сообщением об ошибке не возникает.

6.3. Простейшие приемы ускорения работы с помощью клавиатуры

Настройте свойства формы и компонентов-кнопок (листинг 6.5; см. также рис. 6.2).

Рис. 6.2. Окончательный вид формы Form1 для проекта CALC

Листинг 6.5. Настройка свойств

```
Form1.AcceptButton = button5
button1: Text = &+
button2: Text = &–
button3: Text = &x
button4: Text = &/
button5: Text = &=
```

Результат: кнопка **=** (`button5`) становится кнопкой по умолчанию, эквивалентом ее нажатия является нажатие на клавишу <Enter> (кнопка по умолчанию обводится более толстой рамкой). Символы, указанные на кнопках, подчеркиваются; это является признаком того, что с каждой кнопкой связана клавиша-ускоритель <Alt>+<*подчеркнутый символ*>.

Ошибка: после нажатия на любую кнопку с арифметической операцией все последующие вычисления возвращают значение **0** (поскольку первым символом метки `label1` теперь является `&`, не предусмотренный в операторе `switch`).

Исправление: измените оператор в методе `button1_Click` (см. листинг 6.2) следующим образом:

```
label1.Text = (sender as Button).Text[1].ToString();
```

Это исправление приводит к тому, что в метку копируется только второй символ заголовка (то есть символ с индексом `1`). Поскольку в C# запрещено явно присваивать символьное выражение строковой переменной, полученный символ необходимо преобразовать к типу `string` с помощью метода `ToString`.

6.4. Использование обработчика событий от клавиатуры

С помощью окна **Properties** установите значение свойства `KeyPreview` формы `Form1` равным **`True`**. Определите обработчик события `KeyPress` для формы `Form1` (листинг 6.6).

Листинг 6.6. Обработчик `Form1.KeyPress`

```
private void Form1_KeyPress(object sender, KeyPressEventArgs e)
{
  char c = e.KeyChar;
  switch (c)
  {
    case '+':
      button1_Click(button1, null); break;
    case '_':
      button1_Click(button2, null); break;
    case 'x':
    case '*':
      button1_Click(button3, null); break;
    case '/':
      button1_Click(button4, null); break;
  }
  e.Handled = ! (char.IsDigit(c) || c == ',' || c == '-' || c == '\b');
}
```

Результат: теперь для ввода любой операции достаточно нажать соответствующую клавишу (поскольку клавиша <–> используется при вводе отрицательных чисел, в качестве ускорителя для кнопки `button2` выбрана комбинация <Shift>+<–>, соответствующая символу подчеркивания). При вводе чисел игнорируются все клавиши, кроме цифровых, <–>, <,> и <Backspace> (для обозначения символа, генерируемого клавишей <Backspace>, в C# можно использовать управляющую последовательность `\b`; нажатие этой клавиши обеспечивает удаление символа, расположенного *слева* от курсора в активном поле ввода).

Недочет: в обработчике `Form1_KeyPress` предполагается, что десятичным разделителем является *запятая*, тогда как в системе Windows при других региональных настройках может использоваться разделитель-*точка*.

Исправление: замените в методе `Form1_KeyPress` фрагмент условия

```
c == ','
```

на следующий:

```
c == Application.CurrentCulture.NumberFormat.NumberDecimalSeparator[0]
```

Результат: теперь в качестве допустимого символа используется десятичный разделитель, соответствующий текущим региональным настройкам системы Windows. Для тестирования этой возможности достаточно временно изменить региональные настройки в разделе **Язык и региональные стандарты** Панели управления Windows, выбрав, например, вариант языка **Английский (США)**.

6.5. Контроль за изменением исходных данных

Измените метод `button1_Click` (листинг 6.7), определите обработчик события `TextChanged` для компонента `textBox1` (листинг 6.8) и свяжите полученный обработчик `textBox1_TextChanged` с событием `TextChanged` компонента `textBox2`.

Листинг 6.7. Новый вариант метода `button1_Click`

```
private void button1_Click(object sender, EventArgs e)
{
  label1.Text = (sender as Button).Text[1].ToString();
  label2.Text = "=";
}
```

Листинг 6.8. Обработчик `textBox1.TextChanged`

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
  label2.Text = "=";
}
```

Результат: при изменении операции или содержимого текстовых полей результат предыдущего вычисления стирается (если этого не делать, то старый результат можно ошибочно связать с измененным выражением).

Глава 12. Работа с датами и временем: проект CLOCK

12.1. Отображение на форме текущего времени

После создания проекта CLOCK добавьте в форму `Form1` метку `label1`, а также невизуальный компонент типа `Timer` (этот компонент получит имя `timer1` и будет размещен ниже формы, в области невизуальных компонентов). Настройте свойства формы и добавленных компонентов (листинг 12.1). При задании свойств метки `label1` обратите внимание на ее свойство `Font`, также имеющее набор свойств, два из которых — `Name` и `Size` — надо изменить (в листинге 12.1 для этих свойств используется точечная нотация: `Font.Name` и `Font.Size`). Положение метки `label1` настройте в соответствии с рис. 12.1.

Определите обработчик события `Tick` для компонента `timer1` (листинг 12.2).

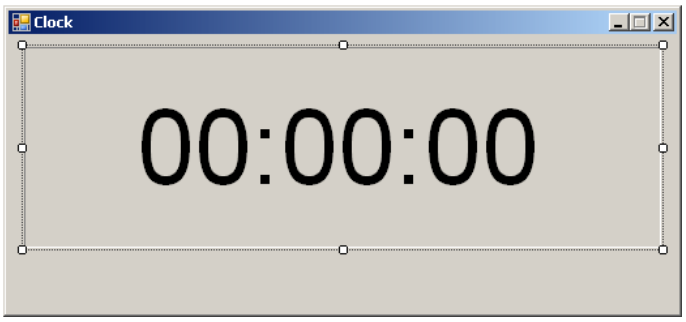

Рис. 12.1. Вид формы Form1 для проекта CLOCK на начальном этапе разработки

Листинг 12.1. Настройка свойств

```
Form1: Text = Clock, MaximizeBox = False,
  FormBorderStyle = FixedSingle,
  StartPosition = CenterScreen
label1: Text = 00:00:00, AutoSize = True,
  TextAlign = MiddleCenter,
  BorderStyle = Fixed3D, Font.Name = Arial,
  Font.Size = 60
timer1: Enabled = True, Interval = 1000
```

Листинг 12.2. Обработчик `timer1.Tick`

```
private void timer1_Tick(object sender, EventArgs e)
{
  label1.Text = DateTime.Now.ToLongTimeString();
}
```

Результат: при работе программы в ее окне отображается текущее время.

Недочет: в течение первой секунды после запуска программы в окне программы сохраняется исходный текст, так как событие `Tick` возникает первый раз только через промежуток времени `timer1.Interval`, равный в нашем случае `1000` (время указывается в миллисекундах).

Исправление: свяжите обработчик `timer1_Click` с событием `Load` формы `Form1`.

Результат: теперь метод `timer1_Click` выполняется первый раз непосредственно перед отображением формы на экране, поэтому в окне сразу отображается правильное время.

12.2. Реализация возможностей секундомера

Разместите в форме `Form1` компонент-*флажок* типа `CheckBox` (он получит имя `checkBox1`) и две кнопки (`button1` и `button2`) и настройте их свойства (листинг 12.3). Положение добавленных компонентов настройте в соответствии с рис. 12.2; для горизонтального центрирования добавленной группы компонентов выделите эти компоненты и нажмите кнопку **Center Horizontally** на панели **Layout**.

В описание класса `Form1` добавьте описание поля:

```
private int t;
```

В начало метода `timer1_Tick` добавьте новые операторы (листинг 12.4).

Определите обработчик события `CheckedChanged` для флажка `checkBox1` и обработчики события `Click` для кнопок `button1` и `button2` (листинг 12.5).

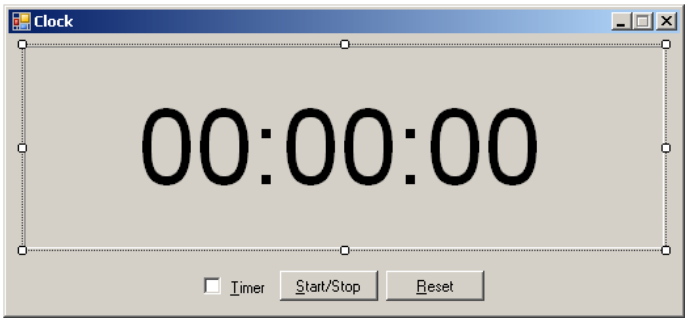

Рис. 12.2. Окончательный вид формы Form1 для проекта CLOCK

Листинг 12.3. Настройка свойств

```
checkBox1: Text = &Timer
button1: Text = &Start/Stop, Enabled = False
button2: Text = &Reset, Enabled = False
```

Листинг 12.4. Новый вариант метода `timer1_Tick`

```
private void timer1_Tick(object sender, EventArgs e)
{
  if (checkBox1.Checked)
  {
    t++;
    label1.Text = string.Format("Timer: {0}:{1}", t / 10, t % 10);
  }
  else
    label1.Text = DateTime.Now.ToLongTimeString();
}
```

Листинг 12.5. Обработчики `checkBox1.CheckedChanged`, `button1.Click` и `button2.Click`

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
  if (checkBox1.Checked)
  {
    t = -1;
    timer1.Interval = 100;
  }
  else
    timer1.Interval = 1000;
  timer1_Tick(this, null);
  button1.Enabled = button2.Enabled = checkBox1.Checked;
  timer1.Enabled = true;
}
private void button1_Click(object sender, EventArgs e)
{
  timer1.Enabled = !timer1.Enabled;
}
private void button2_Click(object sender, EventArgs e)
{
  timer1.Enabled = false;
  t = 0;
  label1.Text = "Timer: 0:0";
}
```

Результат: при установке флажка **Timer** во включенное состояние программа переходит в режим секундомера, причем секундомер сразу запускается, отображая на экране секунды и десятые доли секунд. Запуск и остановка секундомера осуществляются по нажатию кнопки **Start/Stop**, сброс секундомера — по нажатию кнопки **Reset**. Доступны клавиши-ускорители: <Alt>+<T> (смена режима "часы/секундомер"), <Alt>+<S> (запуск/остановка секундомера), <Alt>+<R> (сброс секундомера).

Ошибка: кажущаяся правильность работы секундомера обманчива. В этом можно убедиться, если не останавливать секундомер в течение некоторого времени (выполняя при этом другие действия на компьютере), после чего сравнить результат с точным временем. Причина заключается в том, что событие `Tick` происходит *примерно* каждые 100 мс; кроме того, надо учитывать, что событие `Tick` наступает только при отсутствии других событий, которые требуется обработать программе. Если программа выполняет какой-либо метод длительное время, то в течение этого времени значение секундомера не будет обновляться, а затем отсчет времени продолжится с прежнего значения. Для правильной реализации секундомера надо связать его с *часами компьютера* (используя метод `Now`).

Исправление: в описании класса `Form1` замените строку

```
private int t;
```

на описание новых полей:

```
private DateTime startTime, pauseTime;
private TimeSpan pauseSpan;
```

Назначение добавленных полей следующее: поле `startTime` содержит время начального запуска секундомера; поле `pauseTime` содержит время последней остановки секундомера, а поле `pauseSpan` содержит суммарную длительность всех остановок, выполненных после начального запуска.

Измените методы `timer1_Tick`, `checkBox1_CheckedChanged`, `button1_Click` и `button2_Click` (листинг 12.6).

Листинг 12.6. Новые варианты методов `timer1_Tick`, `checkBox1_CheckedChanged`, `button1_Click` и `button2_Click`

```
private void timer1_Tick(object sender, EventArgs e)
{
  if (checkBox1.Checked)
  {
    TimeSpan s = DateTime.Now - startTime - pauseSpan;
    label1.Text = string.Format("Timer: {0}:{1}", s.Minutes * 60 + s.Seconds, s.Milliseconds / 100);
  }
  else
    label1.Text = DateTime.Now.ToLongTimeString();
}
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
  if (checkBox1.Checked)
  {
    startTime = DateTime.Now;
    pauseSpan = TimeSpan.Zero;
    timer1.Interval = 100;
  }
  else
    timer1.Interval = 1000;
  timer1_Tick(this, null);
  button1.Enabled = button2.Enabled = checkBox1.Checked;
  timer1.Enabled = true;
}
private void button1_Click(object sender, EventArgs e)
{
  timer1.Enabled = !timer1.Enabled;
  if (timer1.Enabled)
    pauseSpan += DateTime.Now - pauseTime;
  else
    pauseTime = DateTime.Now;
}
private void button2_Click(object sender, EventArgs e)
{
  timer1.Enabled = false;
  pauseTime = startTime;
  pauseSpan = TimeSpan.Zero;
  label1.Text = "Timer: 0:0";
}
```

12.3. Альтернативные варианты выполнения команд с помощью мыши

Определите обработчик события `MouseDown` для метки `label1` (листинг 12.7).

Листинг 12.7. Обработчик `label1.MouseDown`

```
private void label1_MouseDown(object sender, MouseEventArgs e)
```

```
{
  if (e.Clicks == 2)
    checkBox1.Checked = !checkBox1.Checked;
  else
  {
    if (!button1.Enabled) return;
    if (e.Button == MouseButtons.Left)
      button1_Click(this, null);
    else
      if (e.Button == MouseButtons.Right)
        button2_Click(this, null);
  }
}
```

Результат: двойной щелчок любой кнопкой мыши на метке приводит к смене режима "часы/секундомер", одинарный щелчок левой кнопкой в режиме секундомера запускает или останавливает секундомер, одинарный щелчок правой кнопкой мыши приводит к сбросу секундомера.

12.4. Отображение текущего состояния часов и секундомера на панели задач

В метод `timer1_Tick` добавьте новый оператор:

```
Text = WindowState == FormWindowState.Minimized ?
  label1.Text : "Clock";
```

Результат: если минимизировать окно приложения CLOCK, то на его кнопке, расположенной на панели задач у нижней границы экрана будет отображаться, в зависимости от режима, текущее время или данные секундомера с пояснением **Timer**. Если окно приложения находится в обычном состоянии, то на кнопке приложения отображается текст, совпадающий с заголовком окна: **Clock**.

Недочет: если минимизировать окно в режиме остановленного секундомера, то текст кнопки приложения не изменится.

Исправление: определите обработчик события `Resize` для формы `Form1` (листинг 12.8).

Листинг 12.8. Обработчик `Form1.Resize`

```
private void Form1_Resize(object sender, EventArgs e)
{
  Text = WindowState == FormWindowState.Minimized ?
    label1.Text : "Clock";
}
```

Результат: теперь текст на кнопке приложения правильно корректируется в любой ситуации, поскольку событие `Resize` происходит не только при изменении размера формы, но и при ее минимизации и возврате в обычное состояние. Кроме того, корректировка текста на кнопке приложения теперь происходит быстрее (до сделанного исправления текст на кнопке изменялся только при очередном срабатывании таймера, а в режиме часов до этого события могла пройти целая секунда).

Примечание

При определении обработчика для события формы `Resize` возникает соблазн связать это событие с уже имеющимся обработчиком `timer1_Tick`, что позволило бы не дублировать одинаковый текст в программе. К сожалению, подобный вариант исправления тоже неправильно работает в режиме остановленного секундомера (объясните почему). Вместе с тем, избежать дублирования текста все же можно, если, наоборот, в обработчике `timer1_Tick` вместо добавленного в данном разделе текста поместить вызов метода `Form1_Resize`: **`Form1_Resize(this, null);`**