# Программирование средств мультимедиа в среде Visual Studio

# на базе технологии Windows Forms (C#)

# Создание и выполнение Windows-проекта

По умолчанию проект содержит класс **Form1** - наследника класса **Form**.

Этот класс содержит точку входа в проект - процедуру **Main**, вызывающую статический метод **Run** класса **Application**, который создает объект класса **Form1** и открывает форму - видимый образ объекта.

Открываемая форма содержит пользовательский интерфейс - окошки, кнопки, списки, другие элементы управления, меню .

Все эти элементы способны реагировать на события, возникающие при выполнении пользователем каких-либо действий - нажатии кнопок, ввода текста, выбора пунктов меню.

# Элементы управления

**Элементы управления** — это компоненты, обеспечивающие взаимодействие между пользователем и программой. Среда Visual Studio.NET предоставляет большое количество элементов, которые можно сгруппировать по нескольким функциональным группам.

**Группа командных объектов**

**Группа текстовых объектов**

**Группа контейнеров**

**Группа графических элементов**

**Диалоговые окна**

**Группа меню**

## Группа командных объектов

Элементы управления **Button, LinkLabel, ToolBar** реагируют на нажатие кнопки мыши и немедленно запускают какое-либо действие. Наиболее распространенная группа элементов.

## Группа текстовых объектов

Большинство приложений предоставляют возможность пользователю вводить текст и, в свою очередь, выводят различную информацию в виде текстовых записей. Элементы **TextBox, RichTextBox** принимают текст, а элементы **Label, StatusBar** выводят ее.

Для обработки введенного пользователем текста, как правило, следует нажать на один или несколько элементов из группы командных объектов.

# Группа переключателей

Приложение может содержать несколько предопределенных вариантов выполнения действия или задачи. Это одна из самых обширных групп элементов, в которую входят **ComboBox, ListBox, ListView, TreeView, NumericUpDown** и многие другие.

# Группа контейнеров

С элементами этой группы действия приложения практически никогда не связываются, но они имеют большое значение для организации других элементов управления, их группировки и общего дизайна формы. Как правило, элементы этой группы, расположенные на форме, служат подложкой кнопкам, текстовым полям, спискам — поэтому они и называются контейнерами. Элементы **Panel, GroupBox, TabControl**, кроме всего прочего, разделяют возможности приложения на логические группы, обеспечивая удобство работы.

# Группа графических элементов

Даже самое простое приложение Windows содержит графику — иконки, заставку, встроенные изображения. Для размещения и отображения их на форме используются элементы для работы с графикой — **Image List, Picture Box**.

# Диалоговые окна

Выполняя различные операции с документом — открытие, сохранение, печать, предварительный просмотр, — мы сталкиваемся с соответствующими диалоговыми окнами. Разработчикам .NET не приходится заниматься созданием окон стандартных процедур: элементы **OpenFileDialog, SaveFileDialog, ColorDialog, PrintDialog** содержат уже готовые операции.

# Группа меню

Многие пользователи настраивают интерфейс приложений на свой вкус: одним нравится наличие определенных панелей инструментов, другим – индивидуальное расположение окон. Но в любом приложении будет присутствовать меню, содержащее в себе доступ ко всем возможностям и настройкам приложения. Элементы MainMenu, ContextMenu представляют собой готовые формы для внесения заголовков и пунктов меню.

# Форма и элементы управления

**Как населить форму элементами управления?** Чаще всего, это делается вручную в режиме проектирования.

Доступные элементы управления, отображаемые на специальной панели (Toolbox), перетаскиваются на форму.

Этот процесс поддерживается особым инструментарием - дизайнером форм (Designer Form).

Как только на этапе проектирования вы сажаете на форму элемент управления, немедленно в тексте класса появляются соответствующие строки кода.

Конечно, все можно делать и программно - появление соответствующих строк кода приводит к появлению элементов управления на форме.

Нужно понимать, что форма - это видимый образ класса Form, а элементы управления, размещенные на форме - это видимые образы клиентских объектов соответствующих классов, наследников класса Control.

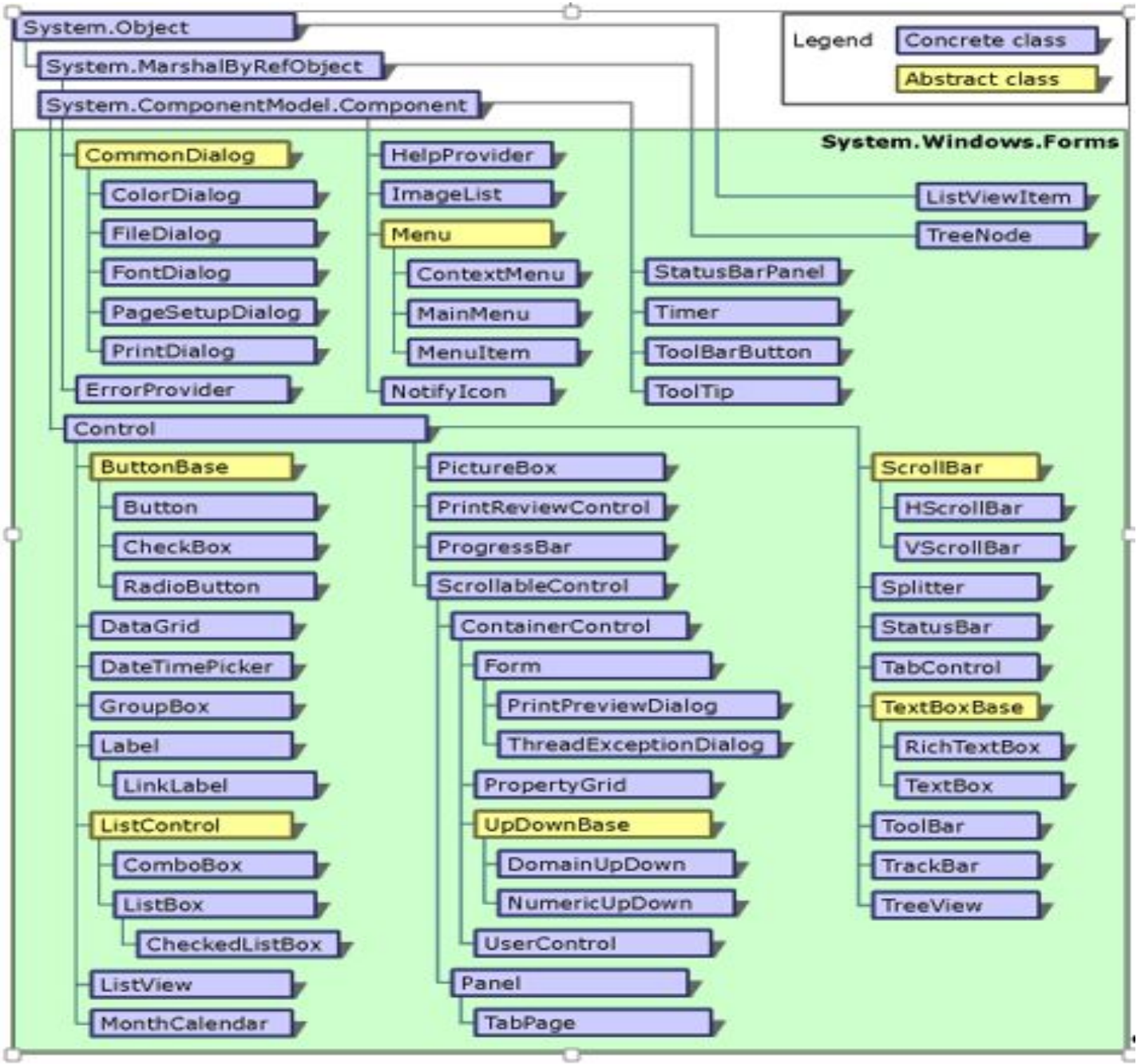
Форма с ее элементами управления есть прямое отражение программного кода.

# Иерархия отношений, связывающих класс Form, класс

Каждый вид элементов управления описывается собственным классом. Библиотека FCL содержит большое число классов, задающих различные элементы управления.

Одним из типов проектов, доступных на C#, является проект, создающий элемент управления, так что ничто не мешает создавать собственные элементы управления и размещать их на формах наряду со встроенными элементами.

Многие фирмы специализируются на создании элементов управления - это один из видов повторно используемых компонентов.



# Взаимодействие форм

Обычное Windows-приложение всегда содержит несколько форм. Одни открываются в процессе работы, другие закрываются.

В каждый текущий момент на экране может быть открыта одна или несколько форм, пользователь может работать с одной формой или переключаться по ходу работы с одной на другую.

Следует четко различать **процесс создания формы** - соответствующего объекта, принадлежащего классу Form или наследнику этого класса, - и **процесс показа формы на экране**.

Для *показа* формы служит метод **Show** этого класса, вызываемый соответствующим объектом.

Для *скрытия* формы используется метод **Hide**.

# Взаимодействие форм

Реально методы Show и Hide изменяют свойство **Visible** объекта, так что вместо вызова этих методов можно менять значение этого свойства, устанавливая его либо в true, либо в false.

**Разница между сокрытием и закрытием формы - между методами Hide и Close**

Метод Hide делает форму невидимой, но сам объект остается живым и невредимым.

Метод Close отбирает у формы ее ресурсы, делая объект отныне недоступным.

Вызвать метод Show после вызова метода Close невозможно, если только не создать объект заново.

Открытие и показ формы всегда означает одно и то же - вызов метода Show. У формы есть метод Close, но нет метода Open.

Формы, как и все объекты, создаются при вызове конструктора формы при выполнении операции new.

# Взаимодействие форм

Форма, открываемая в процедуре **Main** при вызове метода **Run**, называется главной формой проекта.

Ее закрытие приводит к закрытию всех остальных форм и завершению Windows-приложения.

Завершить приложение можно и программно, вызвав в нужный момент статический метод **Exit** класса **Application**.

Закрытие других форм не приводит к завершению проекта.

Зачастую главная форма проекта всегда открыта, в то время как остальные формы открываются и закрываются (скрываются).

Если мы хотим, чтобы в каждый текущий момент была открыта только одна форма, то нужно принять определенные меры, чтобы при закрытии (скрытии) формы открывалась другая. Иначе возможна клинчевая ситуация - все формы закрыты, предпринять ничего нельзя, а приложение не завершено.

Конечно, выход всегда есть - всегда можно нажать магическую тройку клавиш CTRL+ALT+DEL и завершить любое приложение.

# Взаимодействие форм

Можно создавать формы как объекты класса **Form**. Однако такие объекты довольно редки.

Чаще всего создается специальный класс **FormX** - наследник класса **Form**.

Так, в частности, происходит в Windows-приложении, создаваемом по умолчанию, когда создается класс **Form1** - наследник класса **Form**.

Так происходит в режиме проектирования, когда в проект добавляется новая форма с использованием пункта меню **Add Windows Form**.

Как правило, каждая форма в проекте - это объект собственного класса.

Возможна ситуация, когда вновь создаваемая форма во многом должна быть похожей на уже существующую, и тогда класс новой формы может быть сделан наследником класса формы существующей.

# Модальные и немодальные формы

Окно называется **модальным**, если нельзя закончить работу в открытом окне до тех пор, пока оно не будет закрыто.

Модальное окно не позволяет, если оно открыто, временно переключиться на работу с другим окном. Выйти из модального окна можно, только закрыв его.

**Немодальные** окна допускают параллельную работу в окнах.

Форма называется модальной или немодальной в зависимости от того, каково ее окно.

Метод **Show** открывает форму как немодальную, а метод **ShowDialog** - как модальную.

Название метода отражает основное назначение модальных форм - они предназначены для организации диалога с пользователем, и пока диалог не завершится, покидать форму не разрешается.

# Переключение или переход между формами (окнами) c#

**Задача:** Создать 2 формы и переключаться между ними при нажатии кнопки

Создаём  две формы и на них размещаем по одной кнопке и пишем следующие обработчики...

**Обработчик формы 1**

```
public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
         private void button1_Click(object sender, EventArgs e)
        {
            Form2 f2 = new Form2(this);
            this.Hide();
            f2.Show();
        }
    }
```

# Переключение или переход между формами (окнами) c#

**Обработчик формы 1**

```
public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
         private void
button1_Click(object sender, EventArgs e)
        {
            Form2 f2 = new Form2(this);
            this.Hide();
            f2.Show();
        }
    }
```

Здесь при нажатии кнопки на первой форме создаётся экземпляр второй формы и показывается на экран, первая же форма скрывается методом Hide();

**Обратите внимание,** что Form2 вызывается не стандартным конструктором, а конструктором вида

```
Form2 f2 = new Form2(this);
```

*Это делается для того что бы потом можно было вернуться к первой форме. Как аргумент в конструктор формы Form2 форма Form1 передаёт саму себя.*

# Переключение или переход между формами (окнами) c#

## Обработчик формы 2

```
public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }

        private Form1 _f1;
        public Form2(Form1 f1)
        {
            InitializeComponent();
            _f1 = f1;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            _f1.Show();
            this.Close();
        }
    }
```

Здесь прописан тот самый конструктор формы Form2, в котором, как аргумент передаётся форма Form2.

Как член класса прописана форма _f1. При инициализации ей присваивается значение исходной формы f1.

При нажатии на кнопку форма _f1 показывается пользователю, а форма Form2 закрывается методом Close();

**Вывод:** Таким образом мы создаём вторую форму не теряя связи с первой, имея возможность переключаться между окнами (формами) и даже можем передавать туда значения.

# Передача информации между формами

Часто многие формы должны работать с одним и тем же объектом, производя над ним различные операции. Как это реализуется? Обычная схема такова:

Объект создается в одной из форм, чаще всего, в главной.

При создании следующей формы *глобальный объект* передается конструктору новой формы в качестве аргумента.

Одно из полей новой формы должно представлять ссылку на объект соответствующего класса, так что конструктору останется только связать ссылку с переданным ему объектом.

Все это эффективно реализуется, поскольку объект создается лишь один раз, а разные формы содержат ссылки на этот единственный объект.

Если такой глобальный объект создается в главной форме, то можно передавать не объект, требуемый другим формам, а содержащий его контейнер - главную форму.

Это удобнее, поскольку при этом можно передать несколько объектов и не задумываться над тем, какой объект передавать той или иной форме.

Иметь ссылку на главную форму часто необходимо, хотя бы для того, чтобы при закрытии любой формы можно было бы открывать главную, если она была предварительно скрыта.

# Передача информации между формами

Представим себе, что несколько форм должны работать с объектом класса Books.

Пусть в главной форме такой объект объявлен:
public Books myBooks;

В конструкторе главной формы такой объект создается:
myBooks = new Books(max_books);
где max_books - заданная константа.

Пусть еще в главной форме объявлена форма - объект класса NewBook:
public NewBook form2;

При создании объекта form2 его конструктору передается ссылка на главную форму:
form2 = new NewBook(this);

# Передача информации между формами

Класс NewBook содержит поля:

```
private Form1 mainform;

private Books books;
```

а его конструктор следующий код:

```
mainform = form;

books = mainform.myBooks;
```

Теперь объекту form2 доступны ранее созданные объекты, задающие книги и главную форму, так что в обработчике события Closed, возникающего при закрытии формы, можно задать код:

```
private void NewBook_Closed(object sender, System.EventArgs e)
{
    mainform.Show();
}
```

открывающий главную форму.