Практическая работа №7.
Разработка графического интерфейса

Цель работы: научиться разрабатывать графический интерфейс с использованием Windows Forms.

Теоретический материал

Windows Forms – это простейшая технология разработки графического интерфейса на языке программирования C#, основанная на стандартных элементах управления Windows. В основе технологии лежит набор абстракций, описанных в стандартной библиотеке **`System.ComponentModel`**. Эта библиотека используется не только в Windows Forms, но и в других технологиях разработки графического интерфейса: ASP.NET Web Forms, WPF, UWP и MAUI. Эта библиотека состоит из двух частей: абстракций компонентов, из которых строится сам пользовательский интерфейс, и абстракций данных, которые редактируются через любой пользовательский интерфейс. Важнейшей абстракцией данных является описание типов данных для редактирования свойств объектов через графический пользовательский интерфейс. Точкой входа в эту абстракцию является класс **`TypeDescriptor`**, статические методы которого позволяют получать конвертеры и описания свойств и методов объектов. Эти описания используются в дизайнере Windows Forms, но могут использоваться и в пользовательском интерфейсе разработанных приложений.

Важнейшей абстракцией компонента является интерфейс **`IComponent`**, который должны реализовывать классы, предназначенные для настройки их через свойства и события в дизайнере. Этот интерфейс имеет две реализации: класс **`Component`** предназначен для создания компонентов, которые могут управляться из основного домена приложения, сами будучи развёрнуты в дополнительных доменах приложения. Это может быть использовано для создания плагинной архитектуры с выгружаемыми в ходе работы программы плагинами. Класс **`MrashalByValueComponent`** предназначен для взаимодействия доменов приложения путём обмена копиями данных. Класс **`Component`**, как правило, используется для компонентов, в которых происходят какие-то действия: от него наследуются все элементы пользовательского интерфейса (таблицы, кнопки, поля ввода и т.п.). Класс **`MrashalByValueComponent`**, как правило, используется для работы с данными, которые редактируются посредством элементов пользовательского интерфейса.

Принцип работы в Windows Forms состоит в том, чтобы перемещать из панели компонентов компоненты (там отображаются как стандартные компоненты, так и те, которые написаны в ходе разработки приложения) на форму, настраивать их свойства и писать обработчики событий.

Практическая часть

Для начала работы создайте проект приложения Windows Forms (рисунок 1).

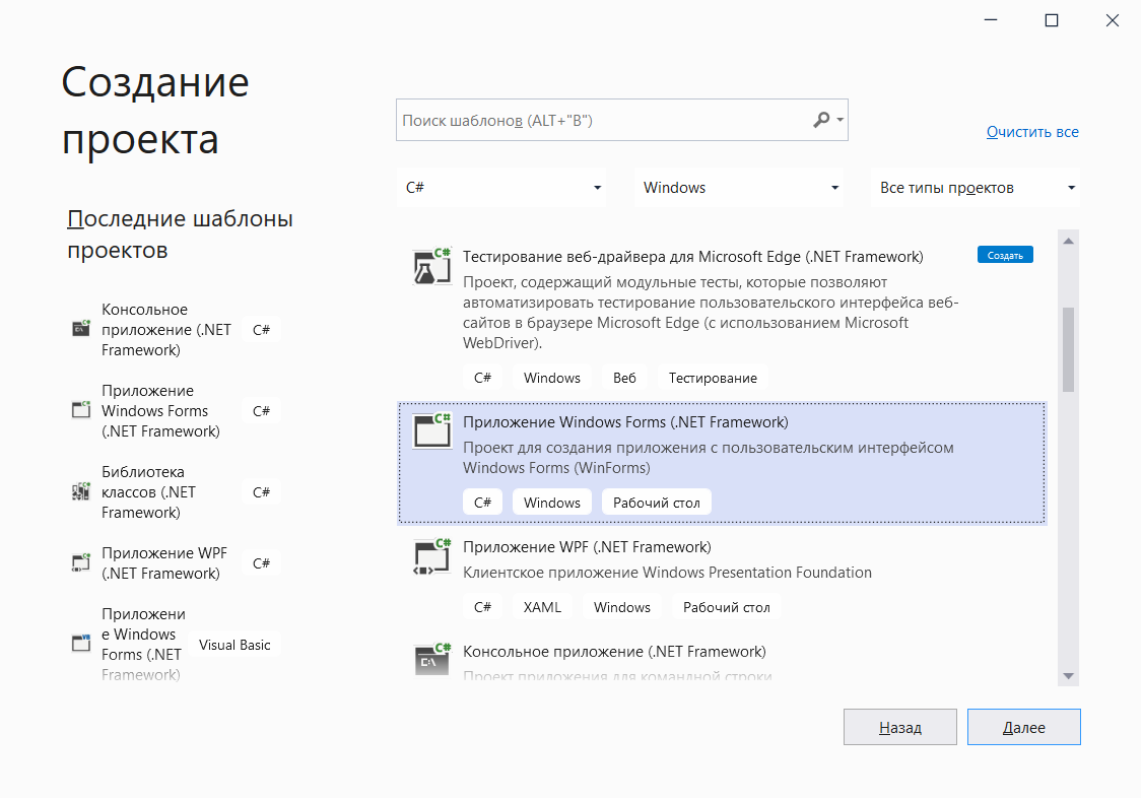

Рисунок 1. Создание проекта приложения Windows Forms.

После этого необходимо выбрать расположение и название проекта (рисунок 2).

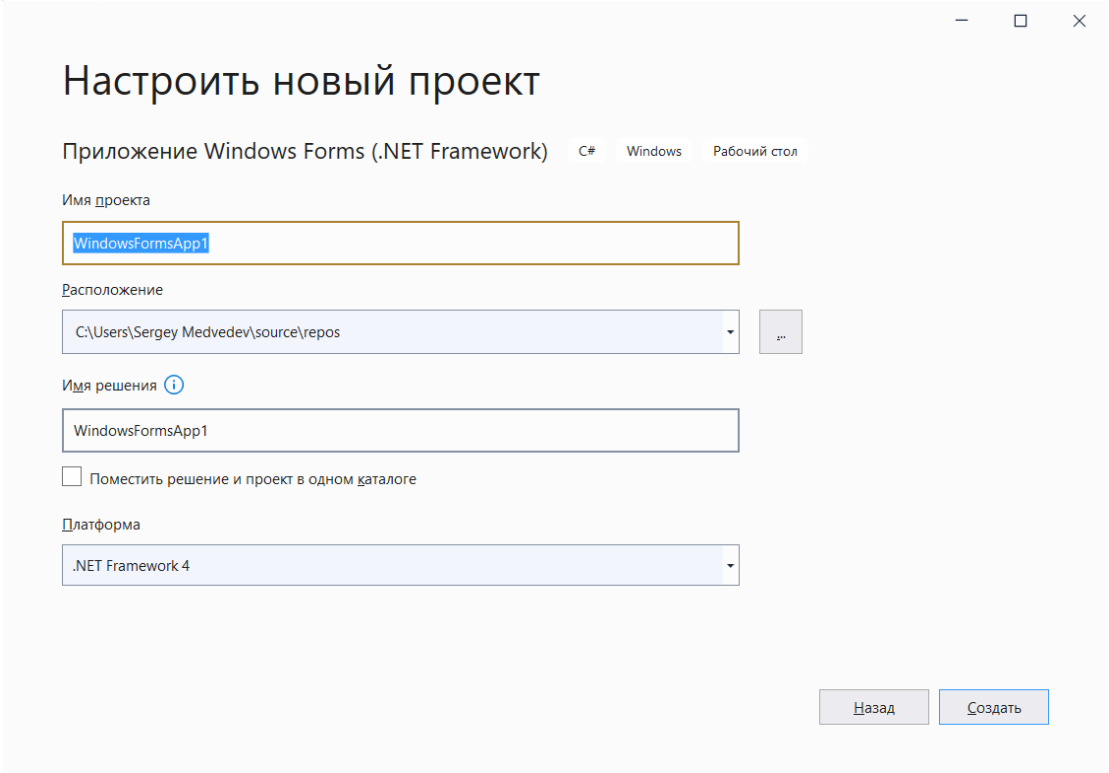

Рисунок 2. Настройка названия и расположения проекта.

После ввода всех начальных настроек создастся проект, содержащий форму **`Form1.cs`** и класс **`Program.cs`**. По умолчанию форма будет открыта в дизайнере и будет выглядеть так, как показано, на рисунке 3.

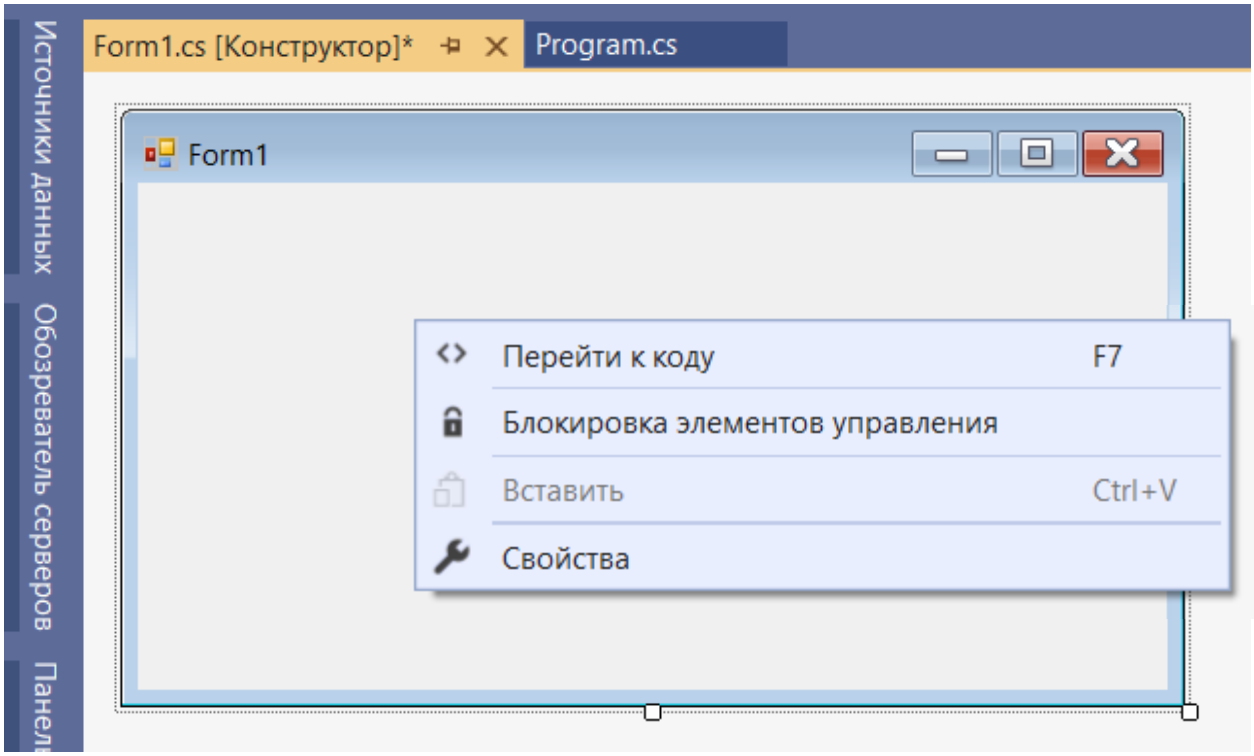

Рисунок 3. Дизайнер форм в Windows Forms.

Чтобы создать элемент управления, откройте панель элементов. На ней будет множество компонентов, сгруппированных по категориям: «Стандартные элементы управления», «Контейнеры», «Меню и панели инструментов», «Данные» и т.д. Чтобы настраивать свойства элементов управления, их нужно выделить на форме и нажать кнопку «Свойства». Свойства выделенного элемента управления отобразятся в панели свойств.

Разработку формы следует начать с разметки. Для этого необходимо использовать контейнеры. Контейнер **`SplitContainer`** позволяет разбить область окна на две части с вертикальной или горизонтальной полосой, которая позволяет менять их соотношение. Свойство **`FixedPanel`** позволяет выбрать, какая из панелей не будет менять размер при изменении размера родительского элемента управления. Свойство **`Orientation`** отвечает за расположение полосы, которая позволяет менять относительный размер панелей: вертикальное или горизонтальное. Второй важнейший контейнер – это **`TableLayoutPanel`**. Он позволяет размещать дочерние элементы управления в виртуальной таблице с возможностью задания ширины столбцов и высоты строк тремя способами: подстраиваться под размер содержимого, фиксированный размер в пикселях или занимать определённый процент свободного пространства. Это можно задать для каждой строки и для каждого столбца этого контейнера.

Для настройки этого контейнера сначала задайте количество строк и столбцов свойствами **ColumnCount** и **RowCount**. Затем войдите в редактор свойств **Rows** и **Columns**, чтобы настроить параметры каждой строки и каждого столбца. Пример настройки четырёх строк показан на рисунке 4. Высота первой строки автоматически подстраивается под её содержимое, вторая строка занимает 20% свободного места, третья 80% свободного места, а четвёртая строка имеет фиксированную высоту 20 пикселей.

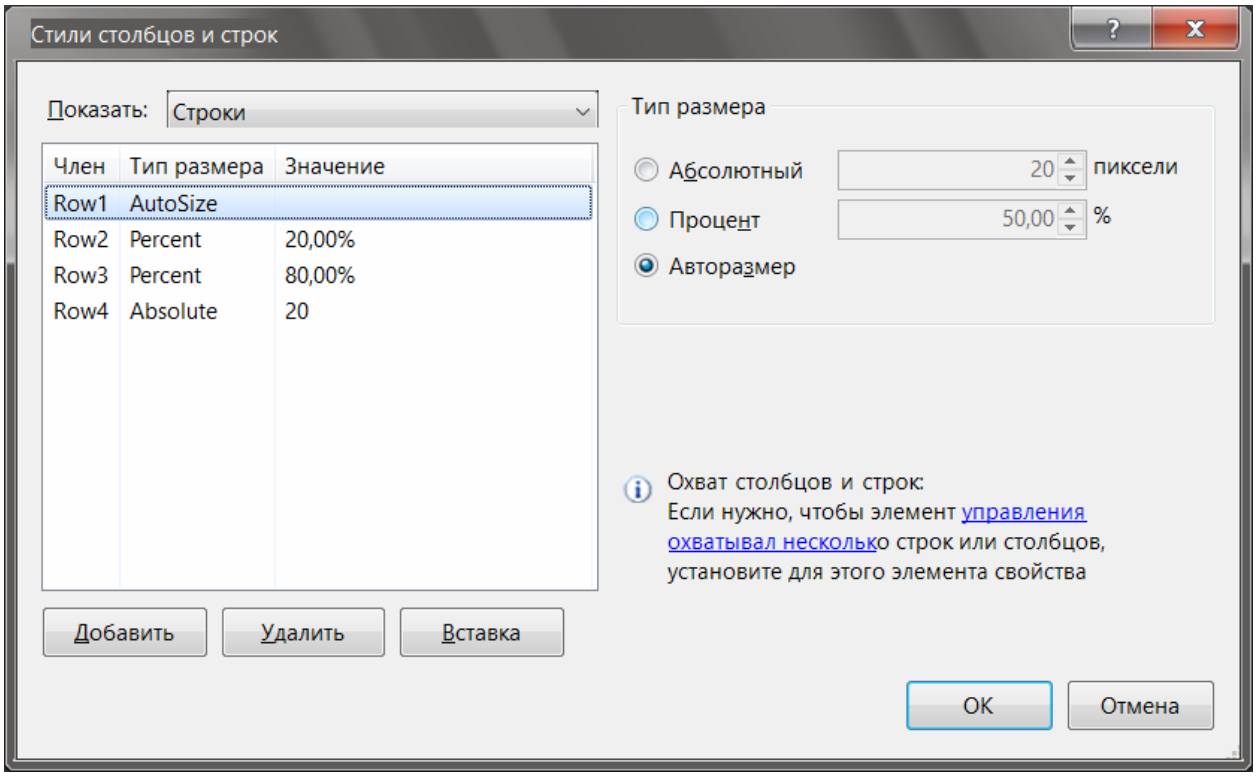

Рисунок 4. Редактор строк в **TableLayoutPanel**.

Чтобы элемент управления сам занимал всё свободное пространство родительского элемента управления, его свойство **Dock** должно иметь значение **Fill**. У контейнера **SplitContainer** это значение по умолчанию, у **TableLayoutPanel** это свойство по умолчанию имеет значение **None**. Скорее всего, потребуется заменить его на **Fill**. Это свойство важно правильно задавать у всех элементов управления, не только у контейнеров.

Чтобы создавать управляющие элементы, необходимо перетаскивать их на форму, перетаскивая их в нужное место контейнеров. Контейнеры при необходимости можно вкладывать друг в друга. Если элемент управления находится внутри **TableLayoutPanel**, то у него появляются дополнительные свойства **ColumnSpan** и **RowSpan**, отвечающие за то, сколько строк и сколько колонок занимает этот элемент управления. Например, на рисунке 5 показано, как диаграмма (**Chart**) может занимать обе колонки **TableLayoutPanel**, в то время как верхняя строка содержит и **Label** с текстом «Подпись», и **TextBox**.

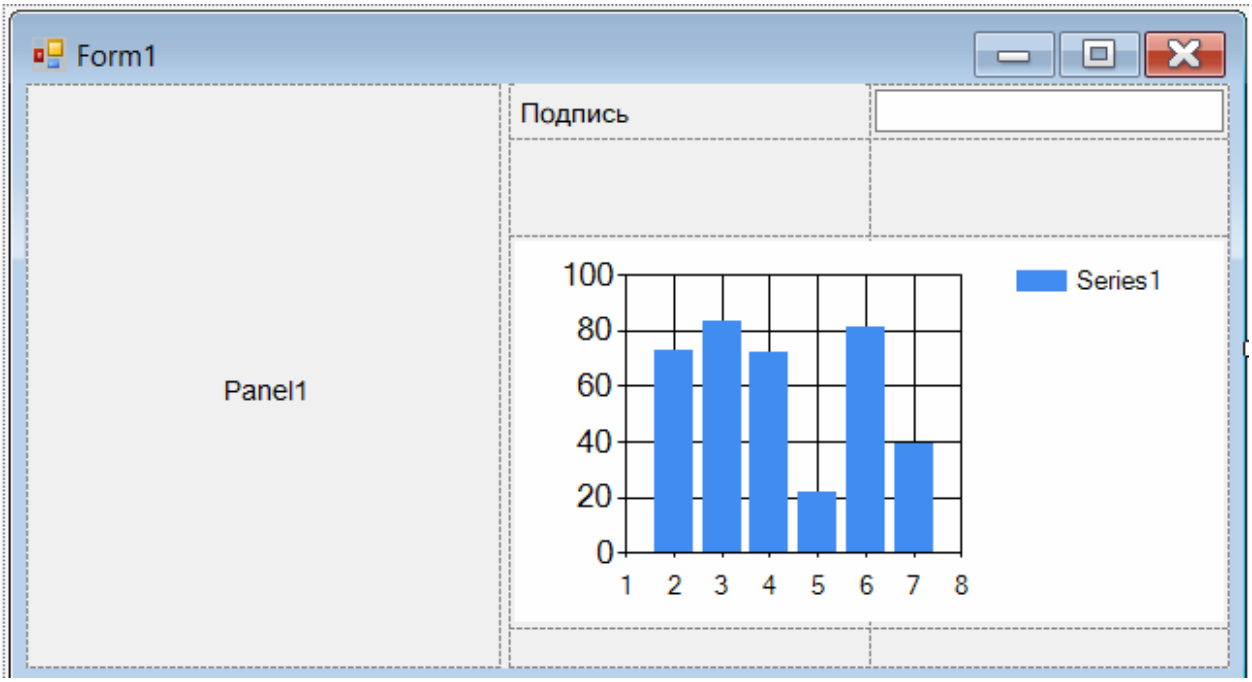

Рисунок 5. Диаграмма, занимающая два столбца **`TableLayoutPanel`**.

Ещё один приём работы с этим контейнером состоит в том, что хотя лучше всего он позволяет управлять расположением элементов на форме, если строки и столбцы имеют либо автоматически подбираемый размер по их содержимому, либо размер, заданный в процентах. Однако если содержимого ещё нет, строки и столбцы с автоматическим размером будут иметь нулевую ширину и высоту. Чтобы решить эту проблему, необходимо сначала задать им фиксированный размер, затем перетащить из панели элементов в нужные ячейки содержимое, а потом заменить режим задания ширины столбца или высоты строки на автоматический. Для управления расстоянием между элементами управления следует использовать свойства **`Margin`** и **`Padding`**. Свойство **`Margin`** есть у всех элементов управления и представляет собой величину их внешних границ в пикселях. Свойство **`Padding`** есть только у контейнеров и представляет величину их внутренних границ (относительно дочерних элементов управления) в пискелях.

Ещё один полезный контейнер – это **`FlowLayoutPanel`**. Этот контейнер позволяет располагать элементы управления в строчку или в столбик, в зависимости от значения свойства **`FlowDirection`**. В этот контейнер можно добавлять сколько угодно элементов управления. Если элементы управления не влезают на сроке, они будут переноситься на следующую строку. Если свойство **`FlowDirection`** имеет значение **`TopDown`** или **`BottomUp`**, то вместо строк элементы управлении будут переноситься в следующий столбик. При изменении размера родительского элемента управления, **`FlowLayoutPanel`** будет при необходимости перемещать все элементы управления, которые требуется переместить, автоматически.

Основной элемент управления для работы с данными – это **`DataGridView`**. Начните настройку элемента с размещения его в нужном контейнере и задания свойств, обеспечивающих корректное поведение этого элемента управления при изменении размеров окна. Для привязки к данным выберите источник данных в свойстве DataSource. Если он ещё не создан, его необходимо создать. Для этого в выпадающем меню в редакторе этого свойства нажмите на кнопку «Добавить источник данных проекта». В появившемся окне выберите «Объект» (рисунок 6).

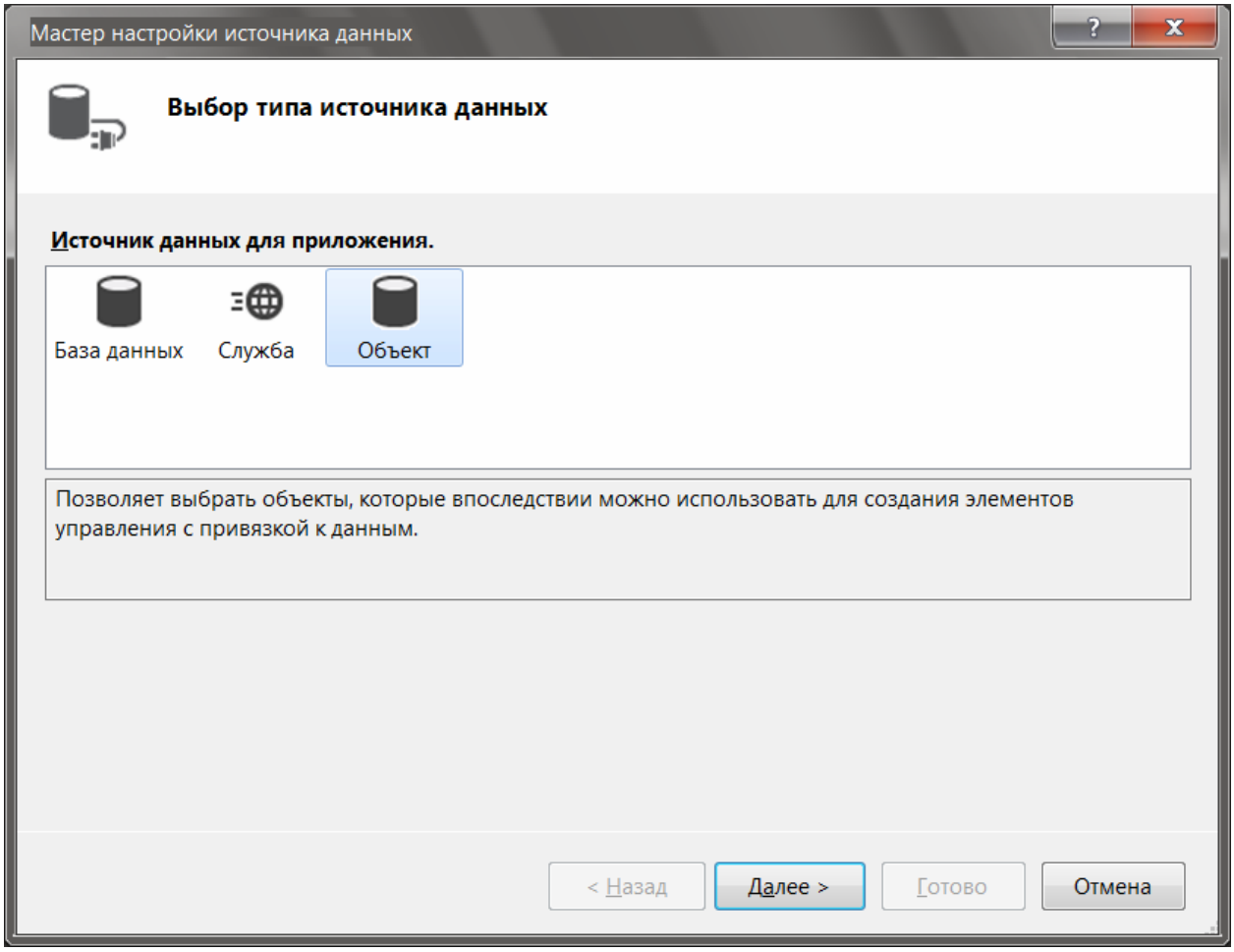

Рисунок 6. Выбор типа источника данных.

На следующем шаге раскройте дерево и поставьте галочку напротив того типа данных (класса), который должен стать источником данных для таблицы. Желательно, чтобы свойства этого класса были размечены атрибутами DisplayName (отображаемое имя колонки) и Browsable(false) (если требуется скрыть какие-нибудь свойства из пользовательского интерфейса). В простейшем варианте этот класс будет просто классом, сгенерированным по таблице для работы с Entity Framework, но в более сложных приложениях необходимо, чтобы классы, которые редактируются через графический интерфейс, были отделены от классов, которые генерируются по базе данных. Пример класса может выглядеть так:

```
using System.ComponentModel;

namespace MyNamespace
{
  public class Goods
  {
    [Browsable(false)]
    public int ID { get; set; }

    [DisplayName("Название")]
    public string Name { get; set; }

    [DisplayName("Описание")]
    public string Description { get; set; }

    [DisplayName("Цена")]
    public double Price { get; set; }
  }
}
```

После нажатия на кнопку «Готово» для этого класса сгенерируются такие колонки таблицы, как показано на рисунке 7.

Рисунок 7. Сгенерировавшиеся колонки таблицы.

Кроме того, при этом в нижней части дизайнера появится компонент **xxxBindingSource** – источник данных. Для того чтобы форма заработала, необходимо нажать клавишу F7, чтобы перейти к коду формы. Дальше после вызова InitializeComponent() необходимо написать следующий код:

```
using (var context = new MyDataContext("some connection"))
    xxxBindingSource.DataSource = context.MyTable.ToList();
```

На место **MyDataContext** и **MyTable** необходимо подставить тот контекст и ту таблицу, которая требуется. По этой таблице должен быть сгенерирован класс, который используется в качестве источника данных для **DataGridView**. К тому же источнику данных, что и **DataGridView**, могут быть привязаны и другие элементы управления. Например, если создать **TextBox**, то через его свойства **(DataBindings)/Text** можно выбрать свойство привязанного класса, например, **Name** (рисунок 8).

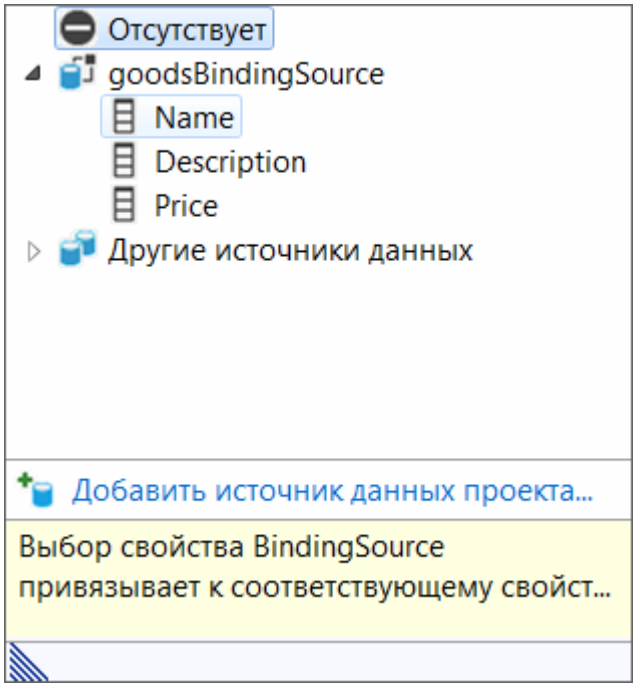

Рисунок 8. Привязка **TextBox** к источнику данных.

В результате с помощью данного текстового поля можно будет просматривать или редактировать выбранную в **DataGridView** строку таблицы.

Для сохранения данных следует сделать кнопку. Элемент управления **Button** находится в категории «Стандартные элементы управления». Перетащите его в нужное место на форме и сделайте двойной клик по нему. Вы перейдёте в редактирование исходного кода формы, и там будет создан метод – обработчик события нажатия кнопки. Чтобы сохранить данные, обратитесь к **xxxBindingSource.DataSource** и сохраните хранящиеся там данные, вызвав метод **SaveChanges** у **DbContext**. Перед созданием обработчика рекомендуется у кнопки задать свойство **(Name)**, чтобы было понятно предназначение кнопки. Значением свойства должен быть допустимых идентификатор C#. В принципе, это рекомендуется делать для любых элементов управления, включая сами формы, перед тем, как создавать обработчики любых событий на этих элементах управления.

Задание

1. Создайте проект приложения Windows Forms.
2. Добавьте в проект классы, сгенерированные кодогенератором по базе данных (желательно из курсового проекта по управлению данными).
3. Разметьте один из сгенерированных классов атрибутами **`System.ComponentModel`** для управления его отображением.
4. На главной форме создайте **`TableLayoutPanel`**.
5. С помощью этого контейнера разместите на форме таблицу (**`DataGridView`**) и кнопку сохранения данных.
6. Привяжите таблицу в пользовательском интерфейсе к одной из таблиц базы данных курсового проекта.
7. Редактирование одной из колонок таблицы сделайте через отдельный элемент (**`TextBox`**, **`ComboBox`** или **`CheckBox`**).
8. Создайте кнопку сохранения данных.
9. Добейтесь того, чтобы приложение позволяло выполнять все действия над данными – чтение, редактирование, добавление и удаление, с сохранением результата в базу данных.

Оформление отчёта

Отчёт должен иметь титульный лист с указанием ФИО и группы студента, а так же темы практической работы. После этого должен быть указан порт и название службы основного сервера, с которым будет вестись работа (скриншот командной строки). Отчёт должен содержать код сгенерированного класса, который требуется отобразить в графическом интерфейсе, с атрибутами **`System.ComponentModel`**. Затем там должен быть код главной формы (конструктор с подключением к базе данных и обработчик кнопки для сохранения). Если для выполнения задания потребовалось сделать ещё какие-нибудь классы, их тоже нужно включить. Необходимо также привести скришоты разработанного приложения, позволяющего редактировать данные.

Примечание. Windows Forms хорошо работает в .NET Framework. Это версии .NET до 5-й не включительно. Однако для .NET Framework последняя поддерживаемая версия Entity Framework – 6. В этой версии рекомендуется при инициализации источника данных использовать метод **`ToBindingList()`**. Начиная с 7 версии, этот метод убрали, потому что решили отвязать Entity Framework от GUI и решили, что такую «графическую» вещь разработчик должен делать самостоятельно. Контекст должен при этом существовать всё время существования формы и уничтожаться в обработчике события **`FormClosed`** методом **`Dispose`**.