

Лабораторная работа №2



ХОД РАБОТЫ



ПРИМЕРЫ



ТЕОРИЯ



ПРИМЕЧАНИЕ



ДОПОЛНИТЕЛЬНОЕ
ЗАДАНИЕ*



ИНТЕРФЕЙС
ПРИЛОЖЕНИЯ



ВОПРОСЫ
ДЛЯ ЗАЩИТЫ

Рисунок 1.1 Обозначения разделов

Лабораторная работа №2

СОДЕРЖАНИЕ

ЗАДАНИЕ	2
ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ*	3
ШАГ 1. СОЗДАНИЕ ПРОЕКТА	5
ОСНОВНЫЕ ЭЛЕМЕНТЫ WINDOWS FORMS	8
ШАГ 2. СОЗДАНИЕ ФОРМЫ	11
ШАГ 3. ДОБАВЛЕНИЕ КЛАССОВ	19
ШАГ 4. РЕАЛИЗАЦИЯ ДОБАВЛЕНИЯ ДАННЫХ В ТАБЛИЦУ	21
ШАГ 5. РЕАЛИЗАЦИЯ ФИЛЬТРАЦИИ (ПОИСКА) ДАННЫХ	21

Лабораторная работа №2

СОДЕРЖАНИЕ

ШАГ 6. СЕРИАЛИЗАЦИЯ, ДЕСЕРИАЛИЗАЦИЯ КЛАССОВ. СОХРАНЕНИЕ В XML	27
ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ	36
СТИЛИЗАЦИЯ ИНТЕРФЕЙСА	36
ФОРМАТИРОВАНИЕ ДАННЫХ	42
ИНТЕРФЕЙС ПРИЛОЖЕНИЯ	44

Лабораторная работа №2

ЗАДАНИЕ

1. Реализовать оконное приложение (Windows Forms).
2. Оконное приложение должно позволять создавать объекты, отображать список созданных объектов в табличной форме и выполнять поиск и удаление.
3. Реализовать сериализацию и десериализацию классов в приложении.
4. Сохранить наследование в классах (как в лабораторной работе №1)
5. Сделать возможность добавления нового объекта для каждого класса
6. Интерфейс приложения должен быть дополнен кнопками для сохранения и загрузки классов из XML файлов.
7. Операции сохранения и загрузки XML файлов должны быть выполнены с использованием методов `async` и `await`.
8. Поиск данных должен выполняться для каждого класса отдельно
9. Реализовать удаление данных (сохраняя верный порядок строк)

Варианты заданий к лабораторной работе №2

В соответствии с вариантом лабораторной работы № 1.

Лабораторная работа №2



ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ*

1. Возможность изменения темы интерфейса

Добавить выбор цветовой схемы (светлая/тёмная тема).

Использовать Flat стиль кнопок для современного вида.

2. Улучшенный поиск

Реализовать фильтрацию данных в DataGridView в реальном времени (например, при вводе текста в TextBox).

Поиск с подсветкой найденных результатов.

3. Контекстное меню

Добавить правый клик на строку в DataGridView, чтобы появлялось меню с опциями (Редактировать / Удалить).

4. Автосохранение

Автоматически сохранять данные в XML при закрытии программы.

Загружать последние данные при запуске.

5. Drag & Drop XML

Сделать так, чтобы XML-файлы можно было просто перетаскивать в окно приложения для загрузки.

Лабораторная работа №2



ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ*

6. Экспорт в другие форматы

Позволить сохранять данные не только в XML, но и в JSON.

7. Статистика

Добавить внизу статус-бар с инфо: "Объектов в базе: X".

8. Логирование ошибок

Если что-то пошло не так (ошибки при загрузке XML), выводить понятное сообщение пользователю.

9. Генерация тестовых данных

Добавить кнопку "Сгенерировать тестовые данные" для удобства тестирования.

10. Горячие клавиши

Например, Ctrl + S для сохранения, Ctrl + O для загрузки, Esc для очистки формы.

Лабораторная работа №2



Шаг 1. Создание проекта

- "Создать проект"
- "Windows forms(Майкрософт)" (Рисунок 1.2)

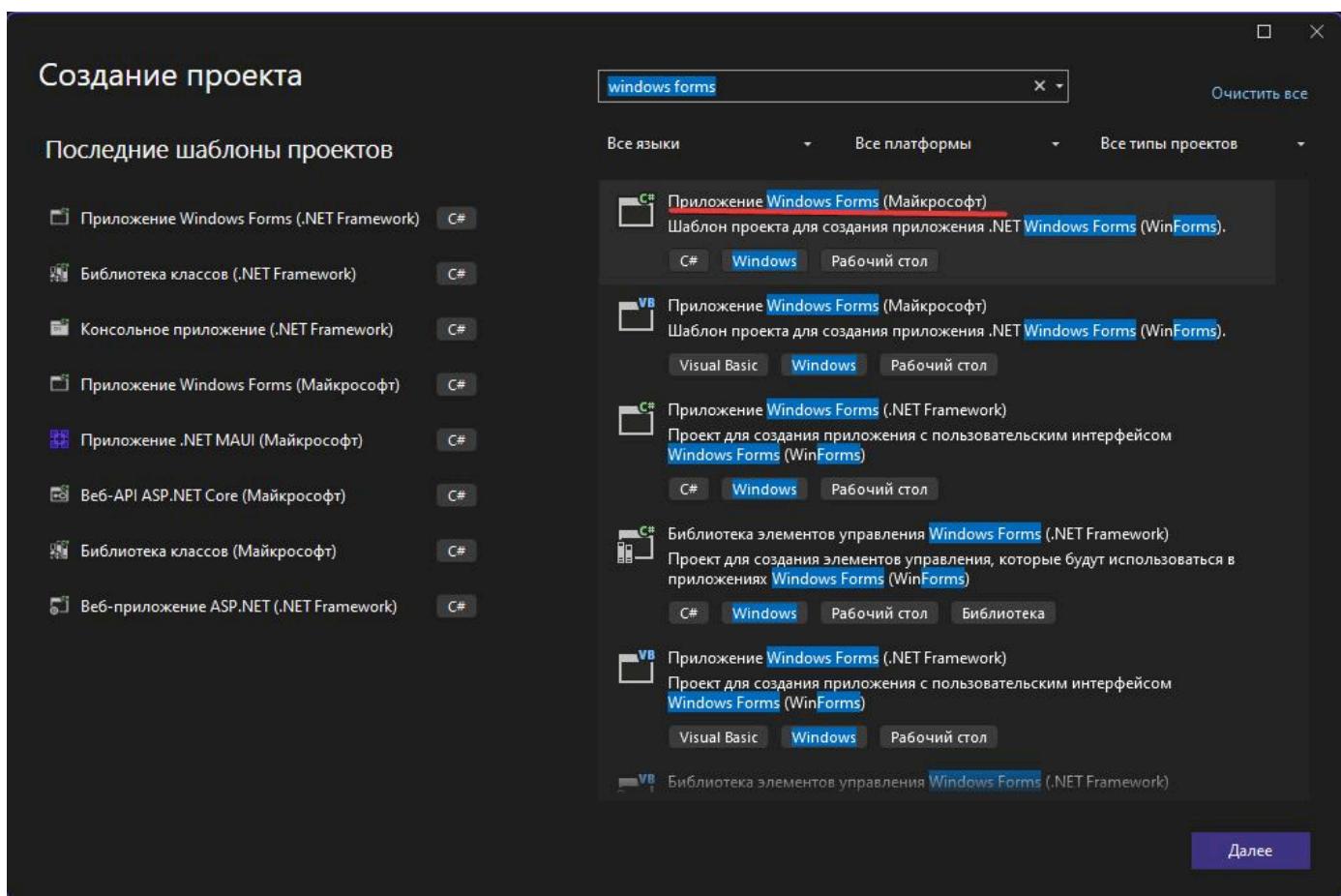


Рисунок 1.2 Обозначения разделов

Лабораторная работа №2



Шаг 1. Создание проекта

- Указываете название формы (содержащее номер лабораторной работы)
- Выбираете последнюю версию платформы из доступных (в примере .NET 8.0) (Рисунок 1.3)

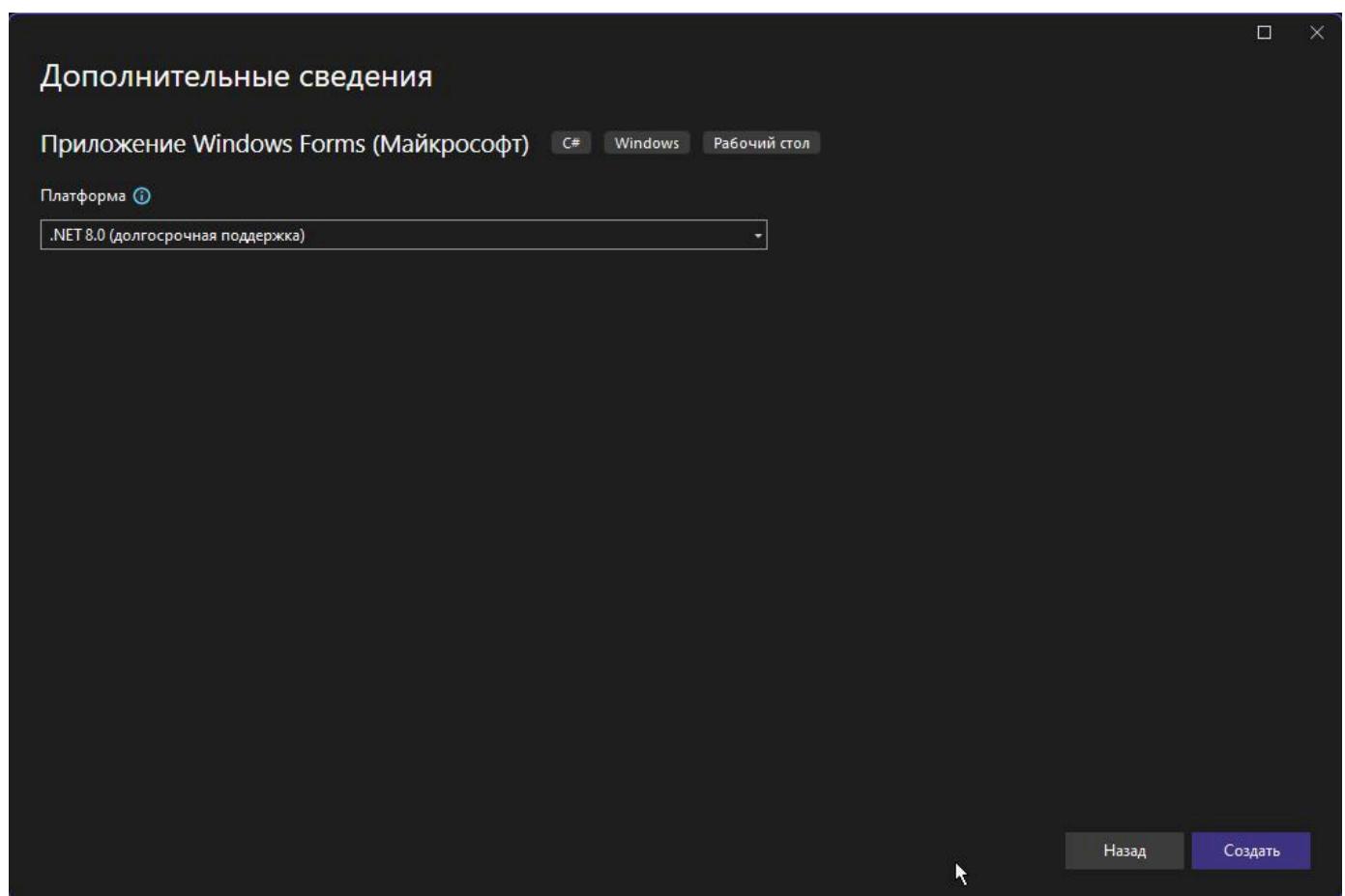


Рисунок 1.3 Настройки проекта

Лабораторная работа №2



Шаг 1. Создание проекта

Перед вами откроется проект с формой по умолчанию

Слева будет “Панель элементов” (если ее нет перейти в “Вид” → “Панель элементов” / Ctrl + Alt + X) (Рисунок 1.4)

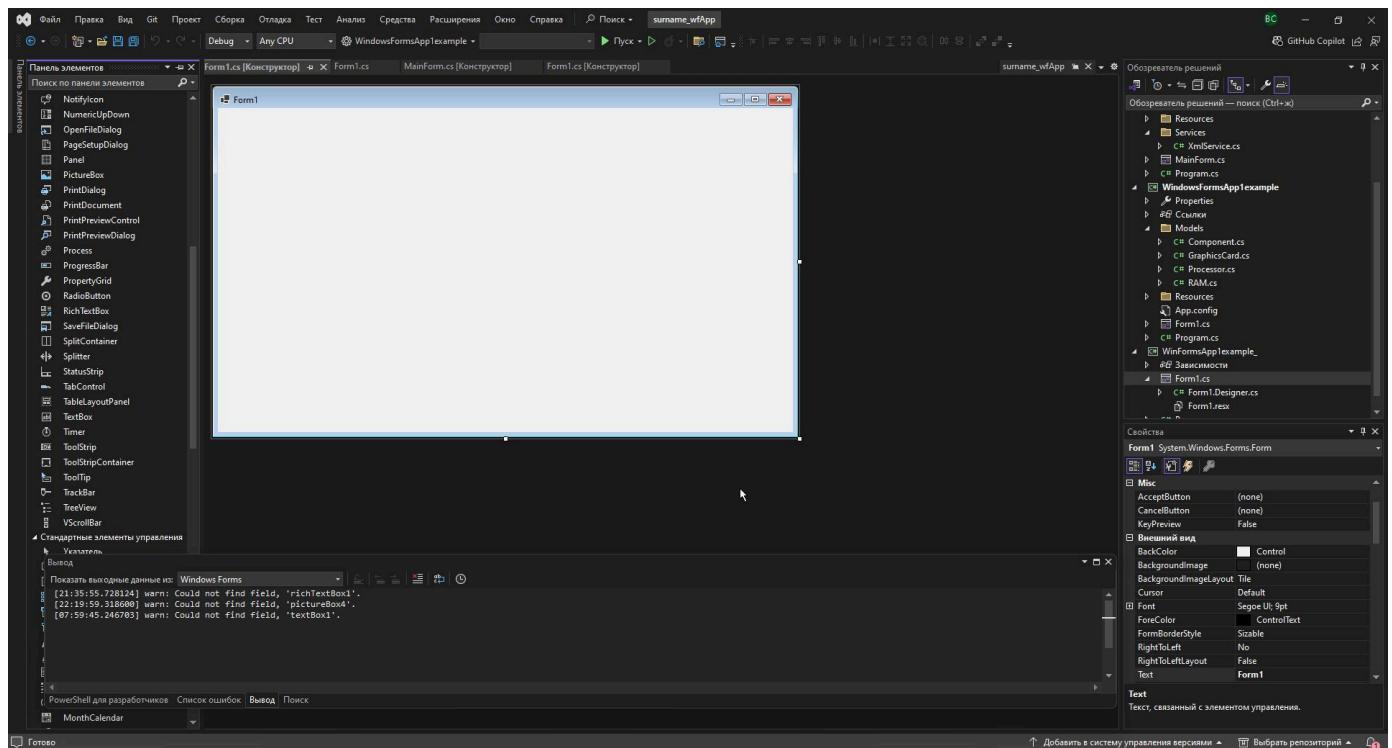


Рисунок 1.4 Окно проекта

Лабораторная работа №2



Основные элементы Windows Forms

Панель элементов содержит множество элементов для работы с формами.

Можно выделить наиболее часто используемые:

Form

Основное окно приложения. Служит контейнером для других элементов управления, предоставляет базовую функциональность для создания пользовательского интерфейса (размещение, масштабирование, обработка событий формы).

Button

Кнопка, по нажатию на которую происходит выполнение заданного кода.

Используется для инициирования действий (например, отправка данных, переход на другую форму).

Label

Элемент для отображения текста. Предназначен для вывода информационных сообщений, описания других элементов и инструкций. Не поддерживает ввод данных.

TextBox

Поле для ввода и отображения однострочного текста. Используется для сбора пользовательского ввода, поиска, ввода паролей (с установкой PasswordChar) и т.д.

RichTextBox

Расширенная версия TextBox, позволяющая форматировать текст (изменять шрифты, цвета, стили). Полезна для редактирования многострочного форматированного текста.

CheckBox

Флажок, позволяющий выбирать или снимать выбор (логическое значение true/false). Применяется в формах для подтверждения, включения настроек и т.п.

Лабораторная работа №2



Основные элементы Windows Forms

ListBox

Список, позволяющий отображать набор элементов, из которых пользователь может выбрать один или несколько. Простой способ представления данных в виде списка.

ComboBox

Комбинированный элемент, который сочетает в себе поле для ввода и выпадающий список. Удобен для выбора одного значения из набора, позволяя пользователю как выбрать из списка, так и ввести значение вручную.

ListView

Многофункциональный список с поддержкой различных видов отображения (иконки, подробности, плитка). Позволяет отображать элементы с подэлементами, а также использовать возможности сортировки и фильтрации (но редактирование ограничено).

DataGridView

Табличное представление данных с поддержкой привязки данных, сортировки, фильтрации и встроенного редактирования ячеек.

PictureBox

Элемент для отображения изображений. Поддерживает различные форматы, позволяет масштабировать или изменять размер изображения в соответствии с настройками.

Panel

Контейнер для других элементов управления, позволяющий группировать связанные элементы и управлять их компоновкой. Может использоваться для создания сложных макетов или динамического отображения/скрытия групп элементов.

Лабораторная работа №2



Основные элементы Windows Forms

Panel

Контейнер для других элементов управления, позволяющий группировать связанные элементы и управлять их компоновкой. Может использоваться для создания сложных макетов или динамического отображения/скрытия групп элементов.

GroupBox

Контейнер с рамкой и заголовком, предназначенный для группировки элементов управления, которые логически связаны между собой. Улучшает визуальную организацию интерфейса.

MenuStrip

Панель меню, располагаемая обычно в верхней части формы. Позволяет создавать выпадающие меню (например, Файл, Правка, Вид и т.д.) для доступа к командам приложения.

ToolStrip

Панель инструментов, содержащая кнопки, текстовые поля, комбобоксы и другие элементы, предназначенные для быстрого доступа к функциям приложения.

StatusStrip

Элемент, расположенный в нижней части формы, для отображения статусной информации (например, индикаторы, сообщения, время работы).

TabControl

Элемент, позволяющий создавать вкладки для организации контента в пределах одного окна. Удобен для разделения функциональности на логически связанные секции.

Лабораторная работа №2



Шаг 2. Создание формы

Для примера будет разработана форма содержащая следующие элементы:

- 1 DataGridView - для добавления элементов и отображения их в табличном виде
- 2 Button - для добавления элементов, удаления, и поиска
- 3 TextBox - для ввода необходимых данных при добавлении элемента или поиске
- 4 PictureBox - для добавления изображения
- 6 Panel - для группировки элементов, создания “вкладок”
- 7 Label - для подписи элементов

Лабораторная работа №2



Шаг 2. Создание и настройка формы

Необходимо разместить соответствующие элементы на форме (пока что схематично). Пример формы изображен на рисунке 2.1.

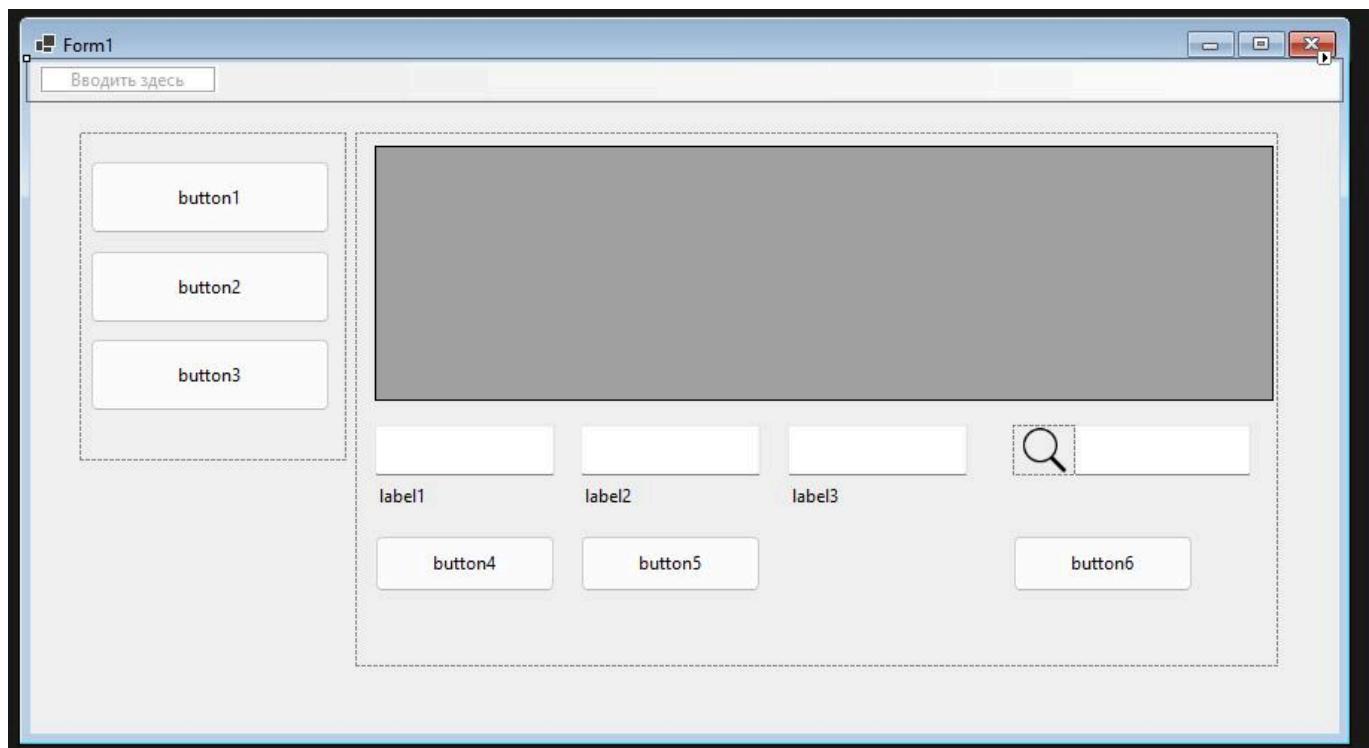


Рисунок 2.1 Пример оформления формы

Лабораторная работа №2



Шаг 2. Создание и настройка формы

Возвращаясь к настройке внешнего вида формы необходимо рассмотреть свойства элементов формы. В правом нижнем углу в проекте можно увидеть окно свойств элемента. Рассмотрим основные свойства, настройка которых потребуется для создания интуитивного и понятного интерфейса, а также изменения внешнего вида элементов.

В верхней части панели есть кнопки для сортировки и перехода по разделам (свойства, эффекты). Для того, чтобы быстрее найти нужное свойство можно выбрать сортировку “по категориям”.

Для того, чтобы изменить название элемента в коде (при обращении к нему, вызове метода) используется свойство (Name). Его можно найти в разделе “Design” (“Разработка”) (Рисунок 2.2).

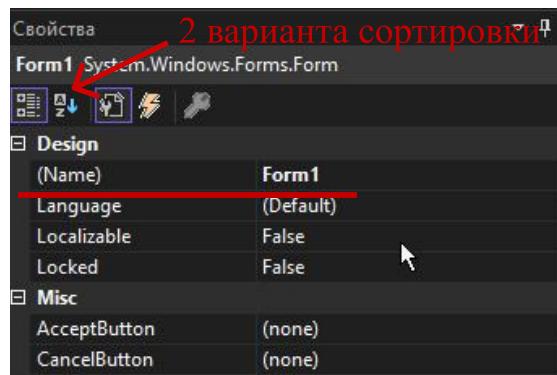


Рисунок 2.2 Свойство Name

Подробно про свойства можно почитать по ссылке:

<https://metanit.com/sharp/windowsforms/2.2.php>

Лабораторная работа №2



Шаг 2. Создание и настройка формы

Для того, чтобы изменить текст, отображаемый на элементе, необходимо указать новое название в свойстве “Text” в разделе “Внешний вид” (Рисунок 2.3)

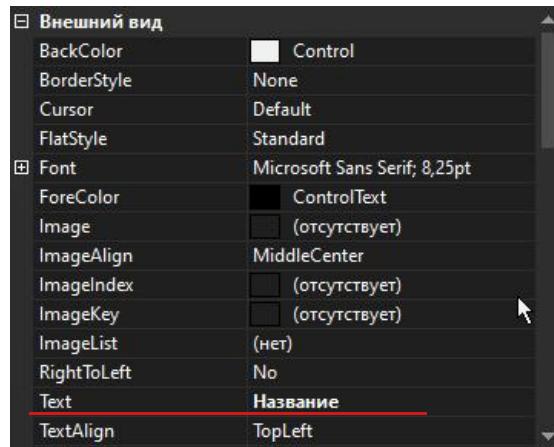


Рисунок 2.3 Свойство Text

Также можно использовать свойство PlaceholderText, чтобы изменить текст, отображаемый на элементе. Свойство находится в разделе “Misc” (Рисунок 2.4). Данное свойство доступно при создании проекта Майкрософт

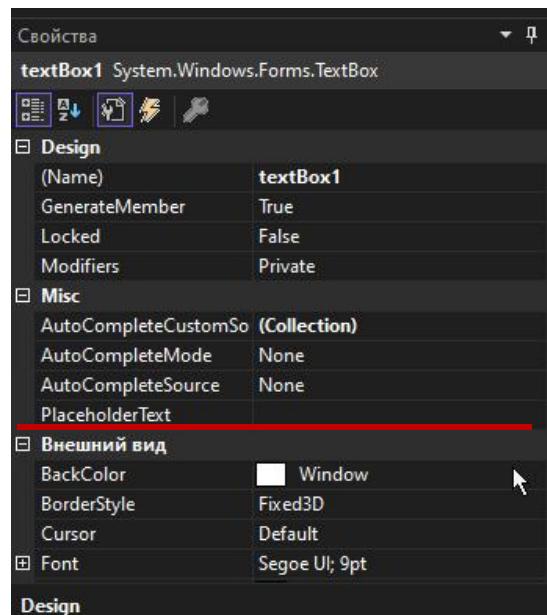


Рисунок 2.4 Свойство Placeholder

Лабораторная работа №2



Шаг 2. Создание и настройка формы

Также необходимо настроить внешний вид элемента TextBox. Для того, чтобы изменить высоту TextBox, необходимо изменить значение параметра Multiline на True (Рисунок 2.5)

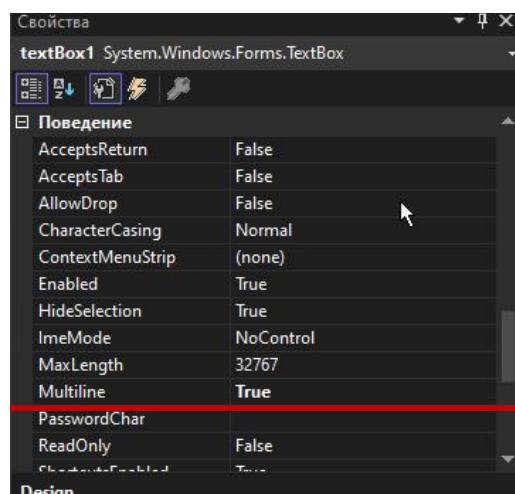


Рисунок 2.5 Свойство Multiline

Для изменения фонового цвета используется свойство BackColor, а для изменения цвета текста - ForeColor (Рисунок 2.6)

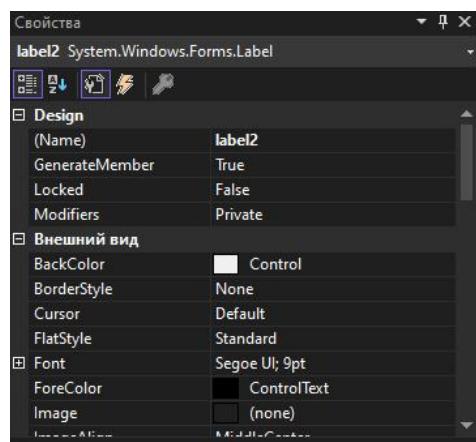


Рисунок 2.6 Свойства для изменения цвета

Лабораторная работа №2



Шаг 2. Создание и настройка формы

Применив необходимые изменения, получим форму, изображенную на рисунке 2.7

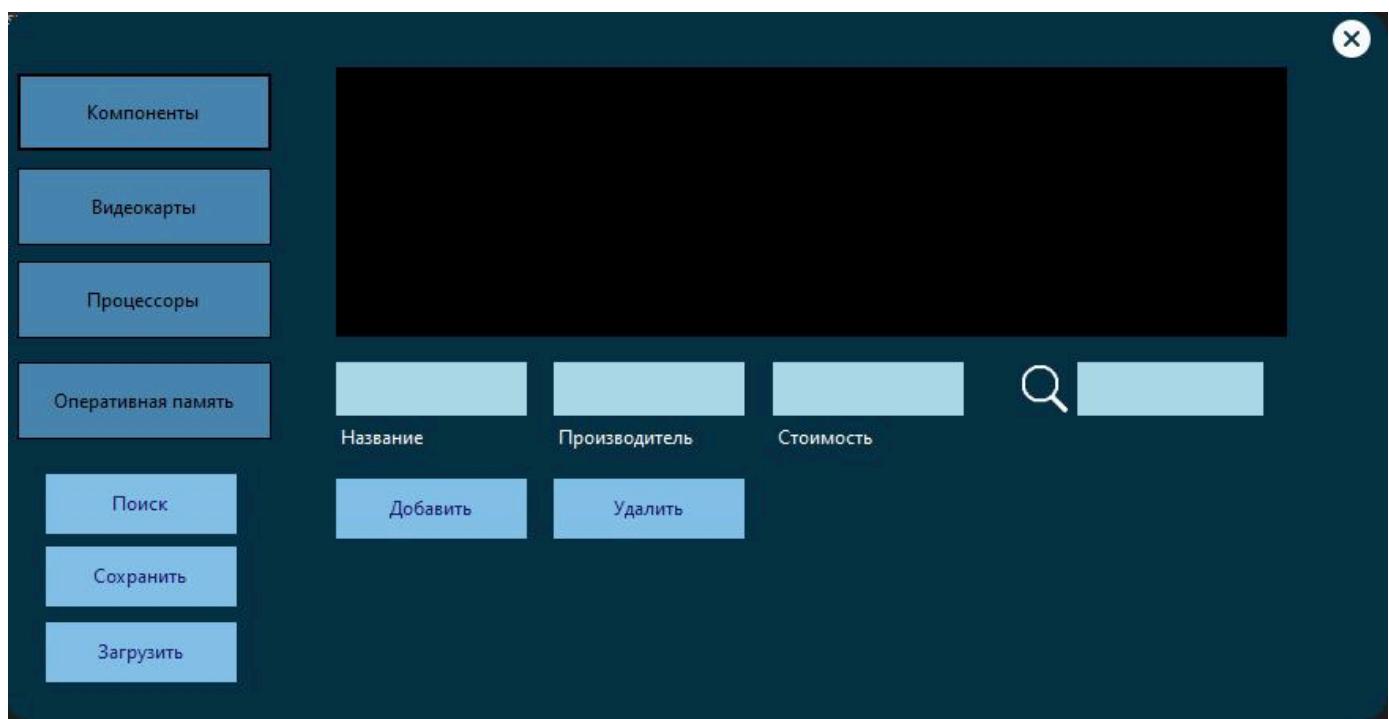


Рисунок 2.7 Пример стилизованной формы

Стилизация интерфейса на ваше усмотрение*

Лабораторная работа №2



Шаг 2. Создание и настройка формы

Важно учесть расположение форм, которые в данном случае находятся друг над другом. Расположение панелей на форме указано на рисунке 2.8.

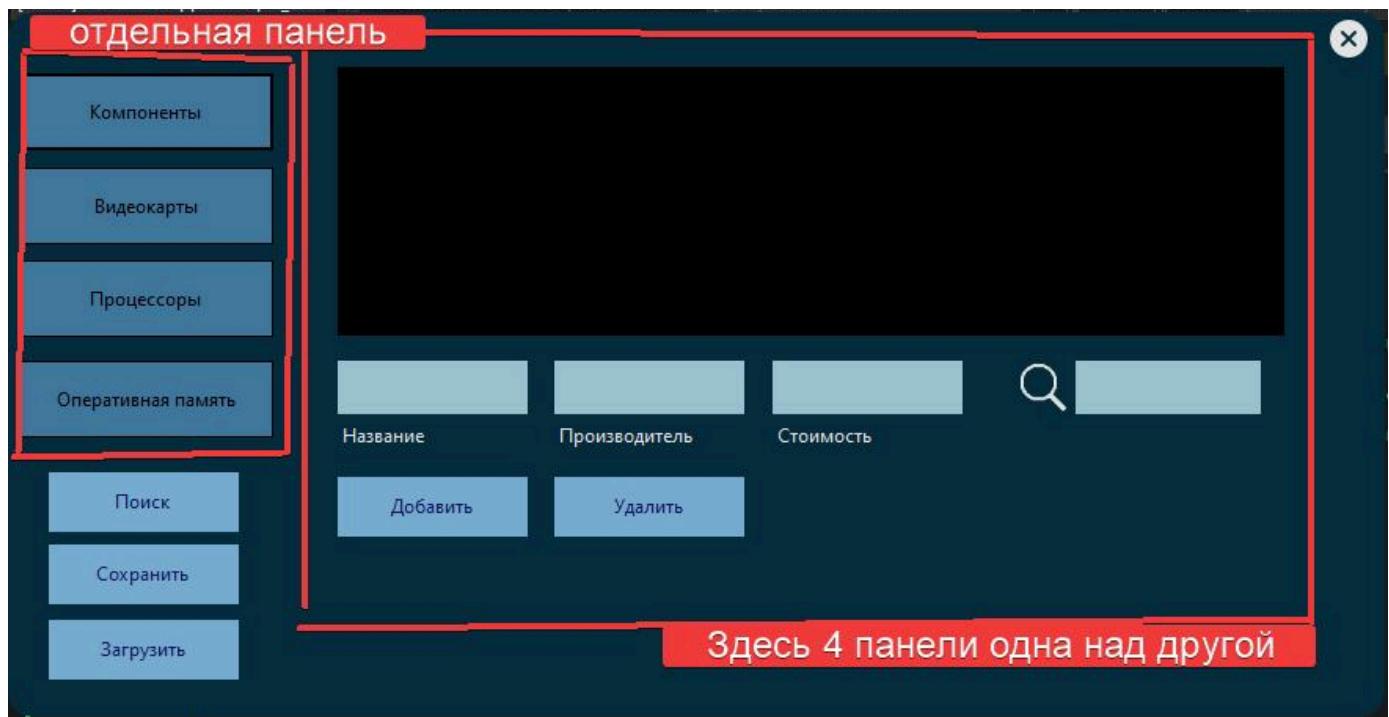


Рисунок 2.8 Структура формы

Также для более детального рассмотрения и управления положением элементов на форме можно воспользоваться окном “Структура документа” (Вид → Другие окна → Структура документа), для этого нужно открыть в файл с самой формой (Form.cs). Структура документа изображена на рисунке 2.9

Лабораторная работа №2



Шаг 2. Создание и настройка формы

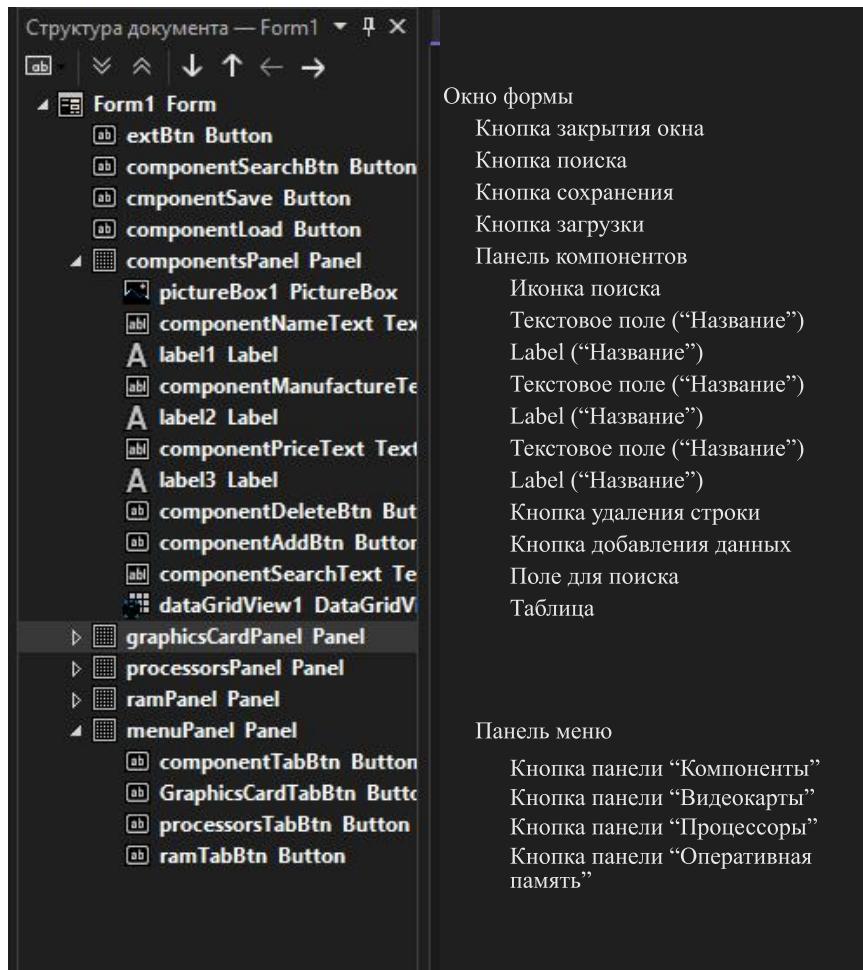


Рисунок 2.9 Структура формы

Также для более детального рассмотрения и управления положением элементов на форме можно воспользоваться окном “Структура документа” (Вид → Другие окна → Структура документа), для этого нужно открыть в файл с самой формой (Form.cs). Структура документа изображена на рисунке 2.9

Лабораторная работа №2



Шаг 3. Добавление классов

Далее необходимо в Решении создать папку и добавить туда классы из лабораторной 1 (Рисунок 3.1)

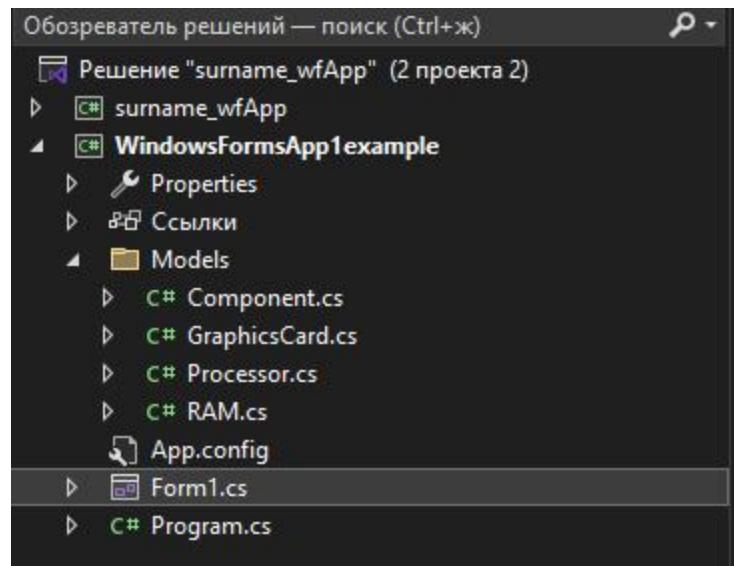


Рисунок 3.1 Создание классов для работы с данными в форме

В каждом классе необходимо наличие 3 свойств, конструктора без параметров (нужен для сериализации) и конструктора с параметрами (Рисунок 3.2).

Лабораторная работа №2



Шаг 3. Добавление классов

```
namespace surname_wfApp.Models
{
    Ссылка 13
    public class Component
    {
        Ссылка 4
        private List<Component> components = new List<Component>();
        Ссылка 4
        public string Name { get; set; }
        Ссылка 4
        public string Manufacturer { get; set; }
        Ссылка 5
        public decimal Price { get; set; } // Только для хранения
        [Browsable(false)] // Скрываем в таблице
        Ссылка 0
        public string FormattedPrice => Price.ToString("N2") + " ₽";

        // Конструктор по умолчанию
        Ссылка 0
        public Component() { }

        Ссылка 3
        public Component(string name, string manufacturer, decimal price)
        {
            Name = name;
            Manufacturer = manufacturer;
            Price = price;
        }
    }
}
```

Рисунок 3.2 Пример структуры класса

Лабораторная работа №2



Шаг 4. Реализация добавления данных в таблицу

После добавления классов необходимо настроить обработчики событий.

В WinForms у элементов управления (кнопок, чекбоксов, текстовых полей и т. д.) есть события (events), которые вызываются при определенных действиях пользователя. Например, клик по кнопке является одним из событий и при настройке появляется в окне свойств (Рисунок 4.1)

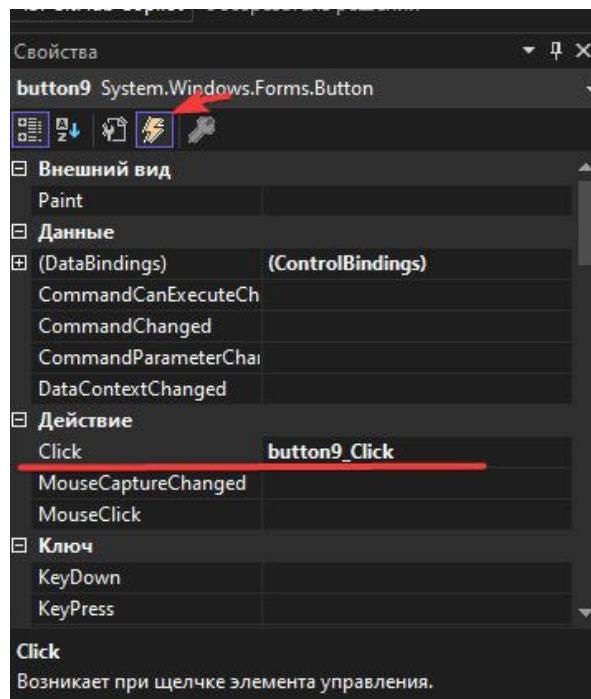


Рисунок 4.1 Добавление событий

Если настраиваемое событие не отрабатывает, необходимо проверить панель свойств на наличие события в данном разделе.

Лабораторная работа №2



Шаг 4. Реализация добавления данных в таблицу

Далее необходимо добавить событие нажатия на кнопку добавления данных в таблицу. Пример метода для добавления данных в форму представлен на рисунке 4.2

```
// Список компонентов (например, процессоры, видеокарты и т.д.)
private List<Component> componentList = new List<Component>();

Ссылка 1
private void componentAddBtn_Click_1(object sender, EventArgs e)
{
    dataGridView1.AutoGenerateColumns = true;

    // Проверяем заполненность полей
    if (string.IsNullOrWhiteSpace(componentNameText.Text) ||
        string.IsNullOrWhiteSpace(componentManufactureText.Text) ||
        string.IsNullOrWhiteSpace(componentPriceText.Text))
    {
        MessageBox.Show("Заполните все поля!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }

    // Преобразуем строковые значения в необходимые числовые типы
    if (!decimal.TryParse(componentPriceText.Text, out decimal price))
    {
        MessageBox.Show("Введите корректное значение цены!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }

    // Создаем новый объект Component и добавляем его в List
    Component newComponent = new Component(componentNameText.Text, componentManufactureText.Text, price);
    componentList.Add(newComponent);

    // Очистка текстовых полей
    componentNameText.Clear();
    componentManufactureText.Clear();
    componentPriceText.Clear();

    // Обновляем DataGridView
    filteredComponentList = null; // Сбрасываем фильтр после добавления
    UpdateDataGrid();
}
```

Рисунок 4.2 - Метод добавления данных в форму

Лабораторная работа №2



Шаг 4. Реализация добавления данных в таблицу

Также необходимо после добавления данных обновить данные в таблице (datagridciew) (Рисунок 4.3)

```
private void UpdateDataGridView()
{
    // Переменные для хранения активного DataGridView и списка данных
    DataGridView activeDataGridView = null;
    IEnumerable<object> activeList = null;

    // Определяем, какой DataGridView и список данных использовать
    if (componentsPanel.Visible) // Панель компонентов
    {
        activeDataGridView = dataGridView1;
        activeList = filteredComponentList ?? componentList;
        // Используем отфильтрованный список, если он существует
    }
    else if (graphicsCardPanel.Visible) // Панель видеокарт
    {
        activeDataGridView = dataGridView2;
        activeList = filteredGraphicsCardList ?? graphicsCardList;
    }
    else if (processorsPanel.Visible) // Панель процессоров
    {
        activeDataGridView = dataGridView4;
        activeList = filteredProcessorsList ?? processorsList;
    }

    // Если активный DataGridView и список определены
    if (activeDataGridView != null && activeList != null)
    {
        // Сбрасываем привязку данных
        activeDataGridView.DataSource = null;

        // Присваиваем обновленный список данных DataGridView
        activeDataGridView.DataSource = activeList.ToList(); // Преобразуем в список, если это не List<T>
    }
    // Очистить все строки, чтобы избежать дублирования данных
    dataGridView4.DataSource = null;

    // Добавляем новые строки вручную для каждого объекта в списке
    foreach (var processor in processorsList)
    {
        // Добавляем строку и указываем значения для каждого столбца
        dataGridView4.Rows.Add(processor.Name, processor.Manufacturer,
            processor.Price, processor.Frequency, processor.Cores);
    }
}
```

Рисунок 4.3 - Метод обновления Datagridview

Можно также добавить событие удаления элемента (строки) из таблицы (Рисунок 4.4)

Лабораторная работа №2



Шаг 4. Реализация добавления данных в таблицу

```
private void componentDeleteBtn_Click(object sender, EventArgs e)
{
    if (dataGridView1.SelectedRows.Count == 0)
    {
        MessageBox.Show("Выберите элемент для удаления!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }

    // Получаем выбранную строку из DataGridView
    DataGridViewRow selectedRow = dataGridView1.SelectedRows[0];
    // Получаем объект Component из DataSource строки
    Component selectedComponent = (Component)selectedRow.DataBoundItem;

    // Удаляем объект из основного списка
    componentList.Remove(selectedComponent);
    filteredComponentList = null; // Сбрасываем фильтр
    UpdateDataGrid(); // Обновляем таблицу
}
```

Рисунок 4.4 - Метод удаления строки из Datagridview

Для добавления данных в заранее созданные поля, необходимо связать название поля с свойствами создаваемого объекта (Рисунок 4.5)

```
private void Form1_Load(object sender, EventArgs e)
{
    // Устанавливаем привязку столбцов к свойствам объекта Processor
    dataGridView4.Columns["Name"].DataPropertyName = "Name";
    dataGridView4.Columns["Manufacturer"].DataPropertyName = "Manufacturer";
    dataGridView4.Columns["Price"].DataPropertyName = "Price";
    dataGridView4.Columns["Frequency"].DataPropertyName = "Frequency";
    dataGridView4.Columns["Cores"].DataPropertyName = "Cores";
}
```

Рисунок 4.5 - Добавления данных в столбцы datagridview без автогенерации

Лабораторная работа №2



Шаг 4. Реализация добавления данных в таблицу

Важно при работе с панелями динамически скрывать и делать их видимыми при выборе соответствующей вкладки. Для этого необходимо создать метод, который будет менять свойство Visible у соответствующей панели при нажатии на кнопку (Рисунок 4.6)

```
Ссылка 4
private void HideAllPanels()
{
    componentsPanel.Visible = false;
    graphicsCardPanel.Visible = false;
    processorsPanel.Visible = false;
    ramPanel.Visible = false;
}

Ссылка 1
private void componentTabBtn_Click(object sender, EventArgs e)
{
    HideAllPanels();
    componentsPanel.Visible = true;
}

Ссылка 1
private void GraphicsCardTabBtn_Click(object sender, EventArgs e)
{
    HideAllPanels();
    graphicsCardPanel.Visible = true;
}

Ссылка 1
private void ramTabBtn_Click(object sender, EventArgs e)
{
    HideAllPanels();
    ramPanel.Visible = true;
}

Ссылка 1
private void processorsTabBtn_Click(object sender, EventArgs e)
{
    HideAllPanels();
    processorsPanel.Visible = true;
}
```

Рисунок 4.6 Метод скрытия панелей

Лабораторная работа №2



Шаг 5. Реализация фильтрации (поиска) данных

Для поиска нужных данных в таблице использует поиск по подстроке в каждом столбце.

```

private void componentSearchBtn_Click(object sender, EventArgs e)
{
    string query = string.Empty; // Объявляем переменную для хранения поискового запроса

    // Определяем активную панель и берём текст из соответствующего поля ввода
    if (componentsPanel.Visible) // Если активна панель компонентов
        query = componentSearchText.Text.Trim().ToLower(); // Получаем текст из поля поиска компонентов,
                                                               // убираем пробелы и приводим к нижнему регистру
    else if (graphicsCardPanel.Visible) // Если активна панель видеокарт
        query = GCSearchText.Text.Trim().ToLower(); // Получаем текст из поля поиска видеокарт,
                                                               // убираем пробелы и приводим к нижнему регистру
    else if (processorsPanel.Visible) // Если активна панель процессоров
        query = processorSearchText.Text.Trim().ToLower(); // Получаем текст из поля поиска процессоров,
                                                               // убираем пробелы и приводим к нижнему регистру
    else // Если ни одна панель не активна
        return; // Завершаем выполнение метода

    // Проверяем, пустой ли запрос
    if (string.IsNullOrEmpty(query)) // Если запрос пустой
    {
        filteredComponentList = null; // Сбрасываем фильтр для компонентов
        filteredGraphicsCardList = null; // Сбрасываем фильтр для видеокарт
        filteredProcessorsList = null; // Сбрасываем фильтр для процессоров
        UpdateDataGrid(); // Обновляем таблицу для отображения полного списка
        return; // Завершаем выполнение метода
    }

    // Выполняем фильтрацию в зависимости от активной панели
    if (componentsPanel.Visible) // Если активна панель компонентов
    {
        filteredComponentList = componentList.Where(comp => // Фильтруем список компонентов
            comp.Name.ToLower().Contains(query) || // Поиск по имени (без учёта регистра)
            comp.Manufacturer.ToLower().Contains(query) || // Поиск по производителю (без учёта регистра)
            comp.Price.ToString().Contains(query) // Поиск по цене (как строке)
        ).ToList(); // Преобразуем результат в список
    }
    else if (graphicsCardPanel.Visible) // Если активна панель видеокарт
    {
        filteredGraphicsCardList = graphicsCardList.Where(gc => // Фильтруем список видеокарт
            gc.Name.ToLower().Contains(query) || // Поиск по имени (без учёта регистра)
            gc.Manufacturer.ToLower().Contains(query) || // Поиск по производителю (без учёта регистра)
            gc.Price.ToString().Contains(query) || // Поиск по цене (как строке)
            gc.MemorySize.ToString().Contains(query) || // Поиск по объёму памяти (как строке)
            gc.Type.ToLower().Contains(query) || // Поиск по типу памяти (без учёта регистра)
            gc.BusWidth.ToString().Contains(query) // Поиск по ширине шины (как строке)
        ).ToList(); // Преобразуем результат в список
    }
    else if (processorsPanel.Visible) // Если активна панель процессоров
    {
        filteredProcessorsList = processorsList.Where(proc => // Фильтруем список процессоров
            proc.Name.ToLower().Contains(query) || // Поиск по имени (без учёта регистра)
            proc.Manufacturer.ToLower().Contains(query) || // Поиск по производителю (без учёта регистра)
            proc.Price.ToString().Contains(query) || // Поиск по цене (как строке)
            proc.Frequency.ToString().Contains(query) || // Поиск по частоте (как строке)
            proc.Cores.ToString().Contains(query) // Поиск по количеству ядер (как строке)
        ).ToList(); // Преобразуем результат в список
    }

    UpdateDataGrid(); // Обновляем таблицу для отображения отфильтрованных данных
}

```

Рисунок 5.1 Метод фильтрации (поиска) данных в таблице

Лабораторная работа №2



Шаг 6. Сериализация, десериализация классов. Сохранение в XML

Сериализация — это процесс преобразования объекта в поток байтов для его сохранения в файле, передаче по сети или хранении в базе данных. Впоследствии этот объект можно восстановить (десериализовать).

Типы сериализации в C#

В C# есть несколько типов сериализации:

1. Binary (Бинарная) — сериализация в двоичный формат (быстро, но неудобно для чтения).
2. XML (Текстовая) — сохраняет объект в формате XML (читаемый формат, но объемный).
3. JSON (Современный стандарт) — сериализация в JSON (удобно для веба, поддерживается большинством языков).
4. Custom (Кастомная) — своя реализация, например, в CSV.

В данном случае будет рассмотрена XML-сериализация

XML (Extensible Markup Language) — это читаемый человеком формат, используемый для хранения структурированных данных.

В C# для работы с XML используется класс XmlSerializer из пространства имен System.Xml.Serialization.

Пример метода сериализации с использованием List приведен на рисунке 6.1

Лабораторная работа №2



Шаг 6. Сериализация, десериализация классов. Сохранение в XML

```

private void SaveDataGridView(DataGridView dgv)
{
    SaveFileDialog saveFileDialog = new SaveFileDialog(); // Создаём объект SaveFileDialog для выбора пользователем пути и имени файла
    saveFileDialog.Filter = "XML Files (*.xml)|*.xml|All Files (*.*)|*.*"; // Устанавливаем фильтр для отображения
                                                                // XML-файлов по умолчанию и опции "все файлы"

    if (saveFileDialog.ShowDialog() == DialogResult.OK) // Открываем диалог сохранения и проверяем, выбрал ли пользователь файл (нажал "OK")
    {
        string filePath = saveFileDialog.FileName; // Получаем полный путь к файлу, выбранному пользователем (например, "C:\data.xml")

        if (dgv == null) // Проверяем, был ли передан действительный DataGridView (не null)
        {
            MessageBox.Show("Нет доступных DataGridView для сохранения."); // Если DataGridView не передан, показываем сообщение об ошибке
            return; // Завершаем выполнение метода
        }

        // Выбираем, какой список сериализовать в зависимости от переданного DataGridView
        if (dgv == dataGridView1) // Если передан DataGridView для компонентов (dataGridView1)
        {
            if (componentList.Count == 0) // Проверяем, содержит ли список componentList данные
            {
                MessageBox.Show("Нет данных для сохранения."); // Если список пуст, показываем сообщение
                return; // Завершаем выполнение метода
            }

            XmlSerializer serializer = new XmlSerializer(typeof(List<Component>)); // Создаём XmlSerializer для сериализации
                                                                // списка List<Component> в XML
            using (TextWriter writer = new StreamWriter(filePath)) // Создаём TextWriter (StreamWriter) для записи данных в файл
            {
                serializer.Serialize(writer, componentList); // Сериализуем componentList в XML и записываем в файл
            } // Автоматически закрываем writer благодаря using
            MessageBox.Show("Данные успешно сохранены."); // Сообщаем пользователю об успешном сохранении
        }
        else if (dgv == dataGridView2) // Если передан DataGridView для видеокарт (dataGridView2)
        {
            if (graphicsCardList.Count == 0) // Проверяем, содержит ли список graphicsCardList данные
            {
                MessageBox.Show("Нет данных для сохранения."); // Если список пуст, показываем сообщение
                return; // Завершаем выполнение
            }

            XmlSerializer serializer = new XmlSerializer(typeof(List<GraphicsCard>)); // Создаём XmlSerializer для сериализации
                                                                // списка List<GraphicsCard> в XML
            using (TextWriter writer = new StreamWriter(filePath)) // Создаём TextWriter для записи данных в файл
            {
                serializer.Serialize(writer, graphicsCardList); // Сериализуем graphicsCardList в XML и записываем в файл
            } // Закрываем writer
            MessageBox.Show("Данные успешно сохранены."); // Сообщаем об успешном сохранении
        }
        else if (dgv == dataGridView4) // Если передан DataGridView для процессоров (dataGridView4)
        {
            if (processorsList.Count == 0) // Проверяем, содержит ли список processorsList данные
            {
                MessageBox.Show("Нет данных для сохранения."); // Если список пуст, показываем сообщение
                return; // Завершаем выполнение
            }

            XmlSerializer serializer = new XmlSerializer(typeof(List<Processor>)); // Создаём XmlSerializer для сериализации
                                                                // списка List<Processor> в XML
            using (TextWriter writer = new StreamWriter(filePath)) // Создаём TextWriter для записи данных в файл
            {
                serializer.Serialize(writer, processorsList); // Сериализуем processorsList в XML и записываем в файл
            } // Закрываем writer
            MessageBox.Show("Данные успешно сохранены."); // Сообщаем об успешном сохранении
        }
    }
}

```

Рисунок 6.1 Пример метода сериализации классов

Лабораторная работа №2



Шаг 6. Сериализация, десериализация классов. Сохранение в XML

Объяснение ключевых моментов

1. SaveFileDialog:

- Используется для взаимодействия с пользователем, чтобы выбрать место и имя файла. Filter ограничивает выбор XML-файлами, но позволяет выбрать "все файлы" (*.*).

2. XmlSerializer:

- Инструмент для преобразования объектов C# (в данном случае списков List<T>) в XML-формат и обратно. Он требует, чтобы классы (Component, GraphicsCard, Processor) имели публичные свойства и пустой конструктор.

3. using (TextWriter ...):

- Обеспечивает корректное закрытие файла после записи, даже если произойдёт ошибка.

4. Проверка на пустой список:

- Перед сериализацией проверяется Count, чтобы не создавать пустой XML-файл.

5. Условные ветви (if-else):

- Код определяет, какой список сериализовать, основываясь на активном DataGridView. Это позволяет сохранять данные только для текущей вкладки.

Лабораторная работа №2



Шаг 6. Сериализация, десериализация классов. Сохранение в XML

Также можно использовать сериализацию данных с помощью DataTable, не используя списки List (Рисунок 6.2)

```
private void SaveDataGridView(DataGridView dgv)
{
    // Создаем и настраиваем диалог сохранения файла
    using (SaveFileDialog saveFileDialog = new SaveFileDialog { Filter = "XML файлы (*.xml)", Title = "Сохранить данные" })
    {
        // Проверяем, нажал ли пользователь кнопку "Сохранить"
        if (saveFileDialog.ShowDialog() == DialogResult.OK)
        {
            // Создаем DataTable с именем "MyTable"
            DataTable dt = new DataTable("MyTable");

            // Добавляем в DataTable колонки из DataGridView
            foreach (DataGridViewColumn column in dgv.Columns)
            {
                dt.Columns.Add(column.Name);
            }

            // Перебираем все строки DataGridView
            foreach (DataGridViewRow row in dgv.Rows)
            {
                // Пропускаем строку, если она новая (пустая)
                if (!row.IsNewRow)
                {
                    // Создаем новую строку для DataTable
                    DataRow dr = dt.NewRow();

                    // Заполняем строку данными из ячеек DataGridView
                    for (int i = 0; i < dgv.Columns.Count; i++)
                    {
                        dr[i] = row.Cells[i].Value ?? DBNull.Value; // Если значение null, заменяем его на DBNull.Value
                    }

                    // Добавляем заполненную строку в DataTable
                    dt.Rows.Add(dr);
                }
            }

            // Сохраняем DataTable в XML-файл с указанием схемы
            dt.WriteXml(saveFileDialog.FileName, XmlWriteMode.WriteSchema);

            // Выводим сообщение об успешном сохранении данных
            MessageBox.Show("Данные сохранены!", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
}
```

Рисунок 6.2 Пример сериализации данных с помощью DataTable

Лабораторная работа №2



Шаг 6. Сериализация, десериализация классов. Сохранение в XML

Десериализация — это процесс обратного преобразования, то есть восстановления объекта из сохраненного состояния (Рисунок 6.3)

```

private void LoadDataGridView(DataGridView dgv) // Метод для загрузки данных из XML-файла в список,
                                              // привязанный к переданному DataGridView
{
    OpenFileDialog openFileDialog = new OpenFileDialog(); // Создаём диалог для выбора файла пользователем
    openFileDialog.Filter = "XML Files (*.xml)|*.xml|All Files (*.*)|*.*"; // Устанавливаем фильтр:
                                                                     // XML-файлы по умолчанию, с опцией "все файлы"

    if (openFileDialog.ShowDialog() == DialogResult.OK) // Открываем диалог и проверяем, выбрал ли пользователь файл
    {
        string filePath = openFileDialog.FileName; // Получаем путь к выбранному файлу (например, "C:\data.xml")

        if (dgv == null) // Проверяем, передан ли действительный DataGridView
        {
            MessageBox.Show("Передан некорректный DataGridView для загрузки данных."); // Если dgv null, показываем ошибку
            return; // Завершаем выполнение
        }

        // Загружаем данные в соответствующий список в зависимости от переданного DataGridView
        if (dgv == dataGridView1) // Если передан DataGridView для компонентов
        {
            XmlSerializer serializer = new XmlSerializer(typeof(List<Component>)); // Создаём сериализатор для List<Component>
            using (TextReader reader = new StreamReader(filePath)) // Открываем поток чтения из файла
            {
                componentList = (List<Component>)serializer.Deserialize(reader); // Десериализуем XML в componentList
            }
            filteredComponentList = null; // Сбрасываем фильтр для отображения полного списка
        }
        else if (dgv == dataGridView2) // Если передан DataGridView для видеокарт
        {
            XmlSerializer serializer = new XmlSerializer(typeof(List<GraphicsCard>)); // Создаём сериализатор для List<GraphicsCard>
            using (TextReader reader = new StreamReader(filePath)) // Открываем поток чтения
            {
                graphicsCardList = (List<GraphicsCard>)serializer.Deserialize(reader); // Десериализуем XML в graphicsCardList
            }
            filteredGraphicsCardList = null; // Сбрасываем фильтр
        }
        else if (dgv == dataGridView4) // Если передан DataGridView для процессоров
        {
            XmlSerializer serializer = new XmlSerializer(typeof(List<Processor>)); // Создаём сериализатор для List<Processor>
            using (TextReader reader = new StreamReader(filePath)) // Открываем поток чтения
            {
                processorsList = (List<Processor>)serializer.Deserialize(reader); // Десериализуем XML в processorsList
            }
            filteredProcessorsList = null; // Сбрасываем фильтр
        }
        else // Если передан неизвестный DataGridView
        {
            MessageBox.Show("Неизвестный DataGridView для загрузки данных."); // Сообщаем об ошибке
            return; // Завершаем выполнение
        }

        UpdateDataGridView(); // Обновляем активный DataGridView с новыми данными
        MessageBox.Show("Данные успешно загружены."); // Сообщаем об успехе
    }
}

```

Рисунок 6.3 Пример метода десериализации

Лабораторная работа №2



Шаг 6. Сериализация, десериализация классов. Сохранение в XML

Разбор метода сериализации

```
ССЫЛКА: 1
private void LoadDataGridView(DataGridView dgv) // Метод для загрузки данных из XML-файла в список,
| // привязанный к переданному DataGridView
```

Рисунок 6.4 Метод десериализации с параметром

Метод принимает параметр dgv типа DataGridView, чтобы знать, в какой DataGridView нужно загрузить данные (например, dataGridView4 для процессоров, dataGridView1 для компонентов или dataGridView2 для видеокарт). Такой подход делает метод универсальным — он может работать с любым DataGridView, а не с конкретным. Это удобно, если у вас несколько таблиц (dataGridView1, dataGridView2, dataGridView4), и вы хотите переиспользовать код.

(Рисунок 6.4)

```
 OpenFileDialog openFileDialog = new OpenFileDialog(); // Создаём диалог для выбора файла пользователем
openFileDialog.Filter = "XML Files (*.xml)|*.xml|All Files (*.*)|*.*"; // Устанавливаем фильтр:
// XML-файлы по умолчанию, с опцией "все файлы"
```

Рисунок 6.5 Объект OpenFileDialog

Создаётся объект OpenFileDialog для выбора файла пользователем. Свойство Filter задаёт фильтр файлов, чтобы в диалоге отображались только XML-файлы по умолчанию, но с возможностью выбрать "все файлы".

Фильтр упрощает пользователю выбор нужного файла, показывая только файлы с расширением .xml.

Формат фильтра "XML Files (*.xml)|*.xml|All Files (*.*)|*.*" — это стандартный синтаксис для OpenFileDialog. Первая часть (XML Files (*.xml)) — это описание, вторая (*.xml) — маска файлов. Аналогично для "All Files". (Рисунок 6.5)

Лабораторная работа №2



Шаг 6. Сериализация, десериализация классов. Сохранение в XML

```
if (openFileDialog.ShowDialog() == DialogResult.OK) // Открываем диалог и проверяем, выбрал ли пользователь файл
{
    string filePath = openFileDialog.FileName; // Получаем путь к выбранному файлу (например, "C:\data.xml")

    if (dgv == null) // Проверяем, передан ли действительный DataGridView
    {
        MessageBox.Show("Передан некорректный DataGridView для загрузки данных."); // Если dgv null, показываем ошибку
        return; // Завершаем выполнение
    }
}
```

Рисунок 6.6 Выбор файла и проверка на корректную таблицу

Метод ShowDialog() (Рисунок 6.6) открывает диалоговое окно выбора файла. Если пользователь выбрал файл и нажал "Открыть", возвращается DialogResult.OK. Если пользователь нажал "Отмена", метод завершится, ничего не делая.

string filePath = openFileDialog.FileName;

Свойство FileName возвращает полный путь к выбранному файлу (например, "C:\data.xml"). Этот путь нужен для чтения данных из файла. Без пути не получится открыть файл для десериализации.

```
if (dgv == null)
{
    MessageBox.Show("Передан некорректный DataGridView для загрузки данных.");
    return;
}
```

Проверяется, что переданный DataGridView (dgv) не равен null. Если dgv равен null, дальнейшая работа с ним вызовет исключение NullReferenceException. Эта проверка защищает от таких ошибок.

Лабораторная работа №2



Шаг 6. Сериализация, десериализация классов. Сохранение в XML

```
// Загружаем данные в соответствующий список в зависимости от переданного DataGridView
if (dgv == dataGridView1) // Если передан DataGridView для компонентов
{
    XmlSerializer serializer = new XmlSerializer(typeof(List<Component>)); // Создаём сериализатор для List<Component>
    using (TextReader reader = new StreamReader(filePath)) // Открываем поток чтения из файла
    {
        componentList = (List<Component>)serializer.Deserialize(reader); // Десериализуем XML в componentList
    }
    filteredComponentList = null; // Сбрасываем фильтр для отображения полного списка
}
```

Рисунок 6.7 Сериализация XML

Далее рассмотрим объект XmlSerializer, позволяющий сохранять данные из таблицы в формате XML (Рисунок 6.7)

Если переданный DataGridView — это dataGridView1 (для компонентов), данные загружаются в componentList.

- XmlSerializer serializer = new XmlSerializer(typeof(List<Component>));
 - Создаётся объект XmlSerializer для типа List<Component>. XmlSerializer знает, как преобразовать XML в список объектов Component.
 - typeof(List<Component>) указывает, что ожидается XML-файл, содержащий список объектов Component.
- using (TextReader reader = new StreamReader(filePath))
 - StreamReader открывает файл по пути filePath для чтения текста.
 - Конструкция using гарантирует, что поток (StreamReader) будет закрыт после использования, даже если произойдёт исключение.
- componentList = (List<Component>)serializer.Deserialize(reader);
 - Метод Deserialize читает XML из потока и преобразует его в объект типа List<Component>.
 - Приведение (List<Component>) необходимо, так как Deserialize возвращает объект типа object.
- filteredComponentList = null;
 - Сбрасывается фильтр, чтобы после загрузки отображался полный список компонентов, а не отфильтрованный.

Лабораторная работа №2



Шаг 6. Сериализация, десериализация классов. Сохранение в XML

Этот блок загружает данные из XML-файла в список componentList, который привязан к dataGridView1. Сброс фильтра (filteredComponentList = null) гарантирует, что пользователь увидит все загруженные данные.

Почему так: Использование XmlSerializer — это стандартный способ десериализации XML в .NET. Проверка dgv == dataGridView1 позволяет методу работать с разными DataGridView и списками.

Также важно связать события кнопок сохранения и загрузки с соответствующими таблицами (Рисунок 6.8)

```
Ссылка 1
private void Form1_Load(object sender, EventArgs e)
{
    // Стилизация GroupBox
    ChangeControlsForeColor(groupBoxGC, Color.White);
    groupBox12.ForeColor = this.BackColor; // Цвет рамки в цвет формы
    groupBoxGC.ForeColor = this.BackColor; // Цвет рамки в цвет формы

    //связывают кнопки сохранения/загрузки с DataGridView, к которому они относятся.
    componentSave.Tag = dataGridView1;
    componentLoad.Tag = dataGridView1;
}
```

Рисунок 6.8 Связь событий и кнопок с помощью Tag

Лабораторная работа №2



Дополнительная информация

Обязательно ли указывать атрибут Serializable?

Нет, атрибут [Serializable] не нужен, если используется DataSet / DataTable + XML для сохранения данных.

Почему Serializable не важен?

Атрибут [Serializable] требуется, если объекты сохраняются с помощью бинарной или JSON-сериализации (например, BinaryFormatter или JsonSerializer).

НО при использовании DataSet.WriteXml(filePath) не нужно указывать атрибут так как :

DataSet.WriteXml(filePath) сохраняет данные в XML без необходимости сериализации класса.

Работает с таблицами (DataTable), а не с объектами конкретного List.

Когда [Serializable] был бы нужен?

- Если объекты List сохраняются в файл (например, JSON, Binary, SOAP).
- Если используется BinaryFormatter или
JsonConvert.SerializeObject(graphicsCardList).

Лабораторная работа №2



Дополнительная информация

Если необходимо добавлять данные в уже созданные ячейки DataGridView, а не создавать новые строки, необходимо обновлять значения ячеек в существующих строках. Это можно сделать несколькими способами:

1. Обращение к ячейкам по индексу строки и столбца

В случае, если в DataGridView уже присутствуют строки, можно обновлять данные так:

```
// Обновление данных в первой строке (индекс 0)  
dataGridView1.Rows[0].Cells[0].Value = "Новое значение";  
dataGridView1.Rows[0].Cells[1].Value = 123;
```

2. Работа с DataTable (если DataGridView привязан к нему)

Если DataGridView использует DataTable как источник данных:

```
// Получаем доступ к DataTable
```

```
DataTable dt = (DataTable)dataGridView1.DataSource; // Изменяем данные в нужной  
строке и столбце dt.Rows[0]["НазваниеСтолбца"] = "Новое значение";
```

После этого DataGridView обновится автоматически.

3. Обновление данных через BindingList<T>

Если DataGridView привязан к BindingList<T>, можно менять объект в списке:

```
// Предположим, нас есть BindingList<MyObject> bindingList[0].SomeProperty =  
"Новое значение"; dataGridView1.Refresh();
```

// Принудительно обновляем DataGridView

Важно:

- Убедись, что строки уже существуют перед обновлением (dataGridView1.Rows.Count).
- Если DataGridView связан с DataTable или BindingList<T>, изменения надо делать в источнике данных.

Лабораторная работа №2



Стилизация интерфейса

Для того, чтобы убрать шапку формы используемую по умолчанию, необходимо установить значение None у свойства FormBorderStyle в разделе “Внешний вид” (Рисунок 6.9)

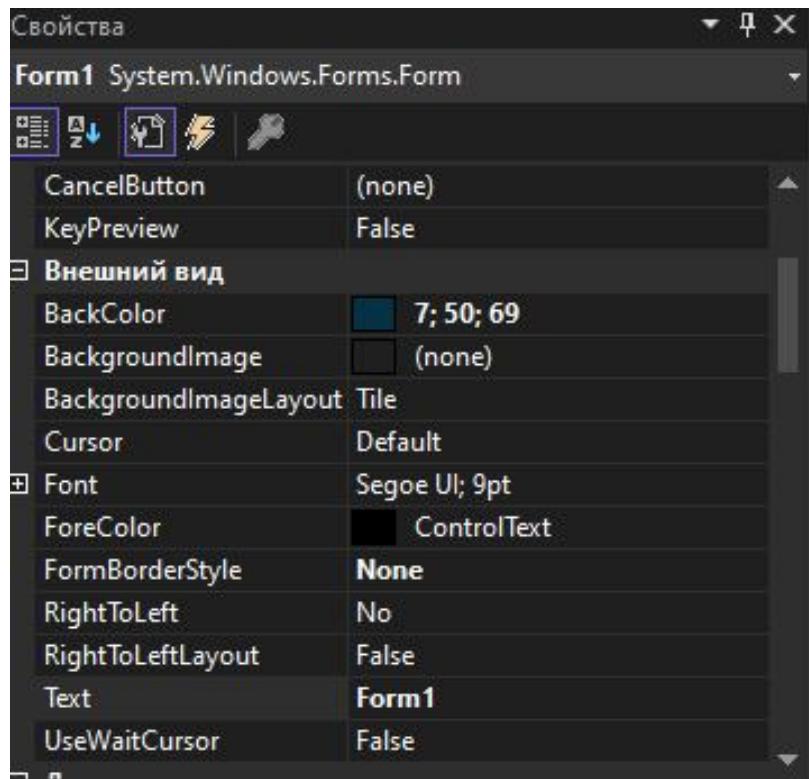


Рисунок 6.9 Свойство FormBorderStyle

После этого возможность перемещать форму с помощью мыши будет отключена. Чтобы вернуть возможность перемещения формы необходимо добавить флаги для реализации перемещения формы, функции для перемещения формы и обработчики событий нажатия мыши. Также можно стилизовать форму закруглив края с помощью функции CreateRoundRectRgn.

Лабораторная работа №2



Стилизация интерфейса

Флаги для реализации перемещения формы и функции для перемещения формы, а также функции для скругления краев формы представлены на рисунке
(Рисунок 6.10)

```
// Флаги для реализации перемещения формы
private bool dragging = false; // Состояние перетаскивания (активно/неактивно)
private Point startPoint = new Point(0, 0); // Начальные координаты клика мыши

// Импорт функции CreateRoundRectRgn для создания закругленных углов формы
[DllImport("Gdi32.dll", EntryPoint = "CreateRoundRectRgn")]
Ссылка 1
private static extern IntPtr CreateRoundRectRgn(
    int nLeftRect,      // X-координата левого верхнего угла
    int nTopRect,       // Y-координата левого верхнего угла
    int nRightRect,     // X-координата правого нижнего угла
    int nBottomRect,    // Y-координата правого нижнего угла
    int nWidthEllipse, // Ширина эллипса для закругления углов
    int nHeightEllipse// Высота эллипса для закругления углов
);

// Импорт функций для перетаскивания формы
[DllImport("user32.dll")]
Ссылка 0
private static extern void ReleaseCapture(); // Сброс захвата мыши

[DllImport("user32.dll")]
Ссылка 0
private static extern void SendMessage(IntPtr hWnd, int msg, int wp, int lp);

private const int WM_NCLBUTTONDOWN = 0xA1; // Сообщение о нажатии кнопки мыши в незакрашенной области окна
private const int HTCAPTION = 0x2; // Идентификатор области заголовка окна
```

Рисунок 6.10 Функции реализации перемещения формы

Лабораторная работа №2



Стилизация интерфейса

Обработчики нажатия мыши приведены на рисунке 6.11

```
// Обработчик события нажатия мыши на форме
Ссылка: 1
private void Form1_MouseDown(object sender, MouseEventArgs e)
{
    // Включаем режим перетаскивания формы
    dragging = true;
    // Запоминаем начальные координаты клика относительно формы
    startPoint = new Point(e.X, e.Y);
}

// Обработчик события движения мыши над формой
Ссылка: 1
private void Form1_MouseMove(object sender, MouseEventArgs e)
{
    // Проверяем, активировано ли перетаскивание
    if (dragging)
    {
        // Преобразуем текущие координаты мыши в экранные координаты
        Point newPosition = PointToScreen(new Point(e.X, e.Y));
        // Обновляем позицию формы с учетом начальной точки клика
        // Это позволяет перемещать форму за точку, где был совершен клик
        this.Location = new Point(
            newPosition.X - startPoint.X,
            newPosition.Y - startPoint.Y
        );
    }
}

// Обработчик события отпускания кнопки мыши
Ссылка: 1
private void Form1_MouseUp(object sender, MouseEventArgs e)
{
    // Выключаем режим перетаскивания
    dragging = false;
}
```

Рисунок 6.11 Обработчики движения мыши

Лабораторная работа №2



Стилизация интерфейса

Также в примере применяется изменение цвета элементов label, размещенных внутри groupBox Рисунок 6.12

```
//Стилизация: смена цвета label внутри groupBox
Ссылка 1
private void ChangeControlsForeColor(System.Windows.Forms.GroupBox groupBox, Color color)
{
    foreach (Control control in groupBox.Controls)
    {
        if (!(control is System.Windows.Forms.TextBox) && !(control is System.Windows.Forms.Button)) // Исключаем TextBox и Button
        {
            control.ForeColor = color;
        }
    }
}

Ссылка 1
private void Form1_Load(object sender, EventArgs e)
{
    // Стилизация groupBox
    ChangeControlsForeColor(groupBoxGC, Color.White);
    groupBox12.ForeColor = this.BackColor; // Цвет рамки в цвет формы
    groupBoxGC.ForeColor = this.BackColor; // Цвет рамки в цвет формы
}
```

Рисунок 6.12 Обработчики движения мыши

Лабораторная работа №2



Форматирование данных

Для того, чтобы отформатировать данные в форме, например автоматически приводить стоимость к денежному формату (добавлять пробел, числа после запятой и знак валюты) необходимо добавить метод, который будет изменять содержимое соответствующего столбца (Рисунок 6.13, Рисунок 6.14)

	Name	Manufacturer	Price	MemorySize	Type	BusWidth
▶	Gh12	China	120 000,00 ₽	16	GDDR6	128

Рисунок 6.13 Форматирование столбца Price

```
private void FormatPriceCell(DataGridView dgv, DataGridViewCellFormattingEventArgs e)
{
    // Проверяем, что индекс корректен
    if (e.ColumnIndex < 0 || e.ColumnIndex >= dgv.Columns.Count)
        return;

    // Проверяем колонку "Price" и наличие значения
    if (dgv.Columns[e.ColumnIndex].Name == "Price" && e.Value != null)
    {
        if (decimal.TryParse(e.Value.ToString(), out decimal price))
        {
            e.Value = price.ToString("N2") + " ₽";
            e.FormattingApplied = true;
        }
    }
}
```

Рисунок 6.14 Метод форматирования строки с стоимостью

Лабораторная работа №2



Форматирование данных

А затем вызывать этот метод у каждой таблицы (свойство CellFromatting) Рисунок 6.15

```
Ссылка 4
private void dataGridView1_CellFormatting(object sender, DataGridViewCellFormattingEventArgs e)
{
    FormatPriceCell(dataGridView1, e);
}

Ссылка 0
private void dataGridView2_CellFormatting(object sender, DataGridViewCellFormattingEventArgs e)
{
    FormatPriceCell(dataGridView2, e);
}

Ссылка 0
private void dataGridView4_CellFormatting(object sender, DataGridViewCellFormattingEventArgs e)
{
    FormatPriceCell(dataGridView4, e);
}
```

Рисунок 6.15 Вызов метода форматирования

Лабораторная работа №2



Интерфейс приложения

Далее представлены примеры интерфеса приложения.

Добавление данных в таблицу Components представлено на рисунке 6.16

	Name	Manufacturer	Price
▶	Processor	AMD	50 000,00 ₽
	GraphicsCard	Nvidia	32 000,00 ₽
	RAM	Kingston	12 000,00 ₽

Название Производитель Стоимость

Поиск Добавить Удалить

Рисунок 6.16 Добавление данных в таблицу компонентов

Удаление осуществляется нажатием на соответствующую строку и затем нажатием кнопки “Удалить” (Рисунок 6.17)

	Name	Manufacturer	Price
▶	Processor	AMD	50 000,00 ₽
	GraphicsCard	Nvidia32	32 000,00 ₽
	RAM	Kingston	12 000,00 ₽

Название Производитель Стоимость

Поиск Добавить Удалить

Рисунок 6.17 Удаление данных из таблицы компонентов

Лабораторная работа №2



Интерфейс приложения

Поиск осуществляется путем ввода данных в поле поиска и нажатием кнопки “Поиск”. Поля поиска в данном случае уникальные для каждой панели (класса), но кнопка поиска одна. (Рисунок 6.18)

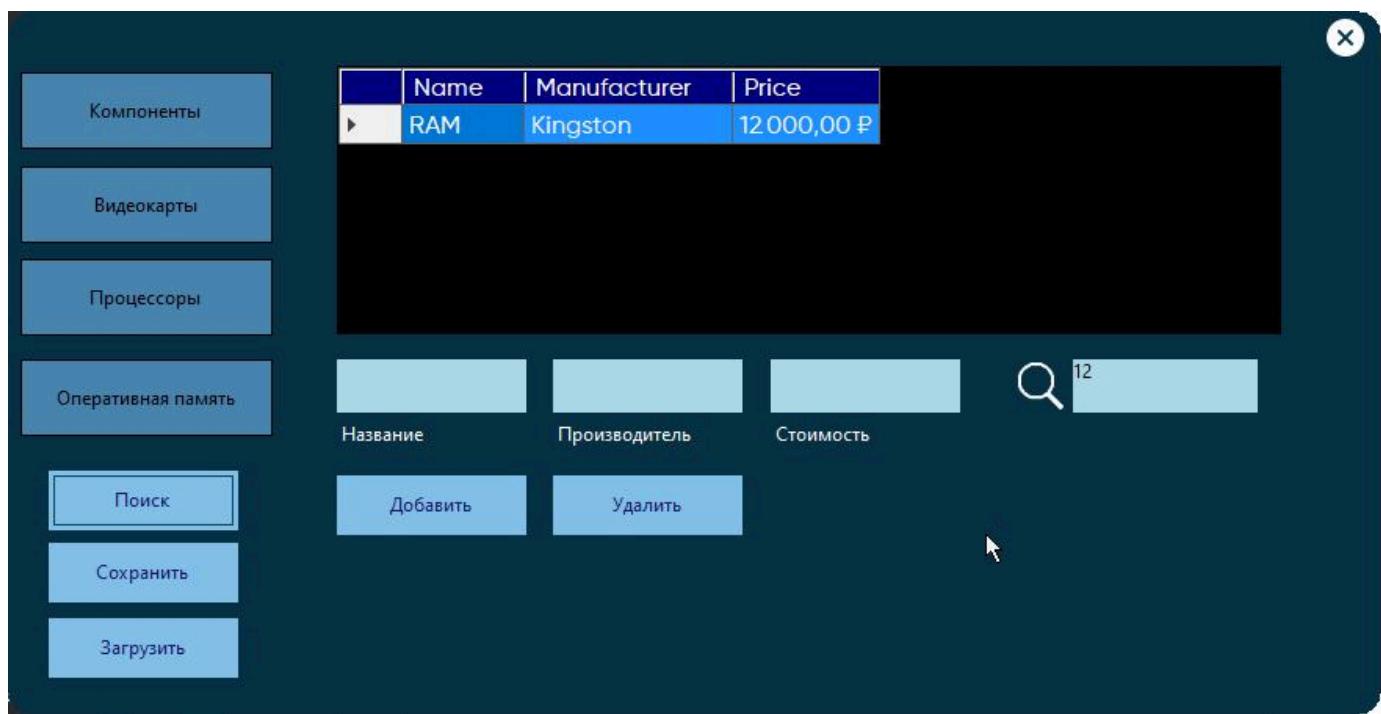


Рисунок 6.18 Добавление данных в таблицу компонентов

Лабораторная работа №2



Интерфейс приложения

Поиск осуществляется путем ввода данных в поле поиска и нажатием кнопки “Поиск”. Поля поиска в данном случае уникальные для каждой панели (класса), но кнопка поиска одна (Рисунок 6.19)

Название	Производитель	Стоимость
Processor	AMD	50 000,00 ₽
RAM	Kingston	12 000,00 ₽

Search bar: 🔍

Buttons: Добавить (Add), Удалить (Delete)

Left sidebar buttons: Компоненты, Видеокарты, Процессоры, Оперативная память, Поиск, Сохранить, Загрузить.

Рисунок 6.19 Добавление данных в таблицу компонентов

Лабораторная работа №2



Интерфейс приложения

Также реализовано добавление данных в таблицы других классов (Рисунок 6.20)

Name	Manufacturer	Price	MemorySize	Type	BusWidth
Palit GeForce R...	Nvida	36000	8	GDDR6	128

Search bar: ×

Filter buttons: Название, Производитель, Стоимость, Объем видеопамяти, Тип видеопамяти, Разрядность шины

Action buttons: Добавить, Удалить

Left sidebar buttons: Компоненты, Видеокарты, Процессоры, Оперативная память, Поиск, Сохранить, Загрузить

Рисунок 6.20 Добавление данных в таблицу видеокарт