

Лабораторная работа №3

Содержание

1 Задание	3
1.1 Дополнительные задания	4
2 Шаг 1. Создание проекта	5
3 Основные элементы Windows Forms	7
4 Шаг 2. Создание формы	9
5 Шаг 3. Добавление классов	15
6 Шаг 4. Реализация добавления данных в таблицу	16
7 Шаг 5. Реализация фильтрации (поиска) данных	21
8 Шаг 6. Сериализация, десериализация классов. Сохранение в XML	22
9 Дополнительная информация	29
10 Стилизация интерфейса	31
11 Форматирование данных	32
12 Интерфейс приложения	33
13 Вопросы для защиты	34

Листинг

5.1 Пример структуры класса	15
6.1 Метод добавления данных в форму	17
6.2 Метод обновления DataGridView	18
6.3 Метод удаления строки из DataGridView	19
6.4 Добавления данных в столбцы DataGridView без автогенерации	19
6.5 Метод скрытия панелей	20
7.1 Метод фильтрации (поиска) данных в таблице	21

8.1	Пример сериализации данных с помощью <code>DataTable</code>	24
8.2	Пример метода десериализации ¹	25
8.3	Метод десериализации с параметром	26
8.4	Объект <code>OpenFileDialog</code>	26
8.5	Выбор файла и проверка на корректную таблицу	26
8.6	Сериализация XML	27

Лабораторная работа №3

1 Задание ²

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)

²Варианты заданий в соответствии с вариантом лабораторной работы № 1

1.1 Дополнительные задания ³

Возможность изменения темы интерфейса

- Добавить выбор цветовой схемы (светлая или тёмная тема).
- Использовать Flat стиль кнопок для современного вида.

Улучшенный поиск

- Реализовать фильтрацию данных в DataGridView в реальном времени (например, при вводе текста в TextBox).
- Поиск с подсветкой найденных результатов.

Контекстное меню

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

Автосохранение

- Автоматически сохранять данные в XML при закрытии программы.
- Загружать последние данные при запуске.

Drag & Drop

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

Экспорт в другие форматы

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

Статистика

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

Логирование ошибок

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

Генерация тестовых данных

- Добавить кнопку "Сгенерировать тестовые данные" для удобства тестирования.

Горячие клавиши

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

2 Шаг 1. Создание проекта ✨

- "Создать проект"
- "Windows forms(Майкрософт)"(Рисунок 2.1)

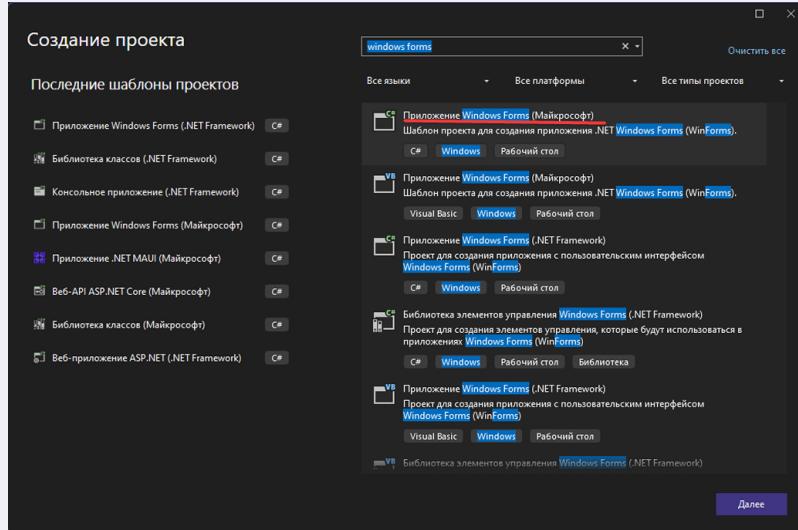


Рисунок 2.1 – Обозначения разделов

- Указываете название формы (содержащее номер лабораторной работы)
- Выбираете последнюю версию платформы из доступных (в примере .NET 8.0) (Рисунок 2.2)

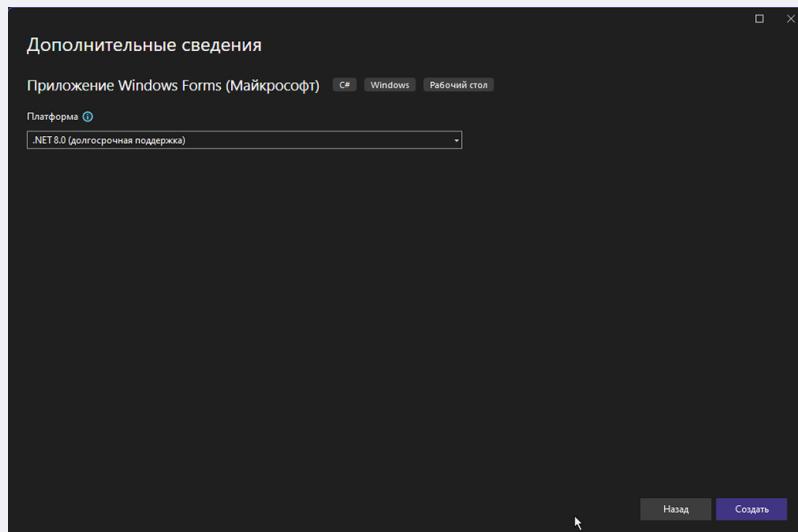


Рисунок 2.2 – Обозначения разделов

³Дополнительные задания выполняются по желанию и не подлежат проверке на зачёте.

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 2.3)

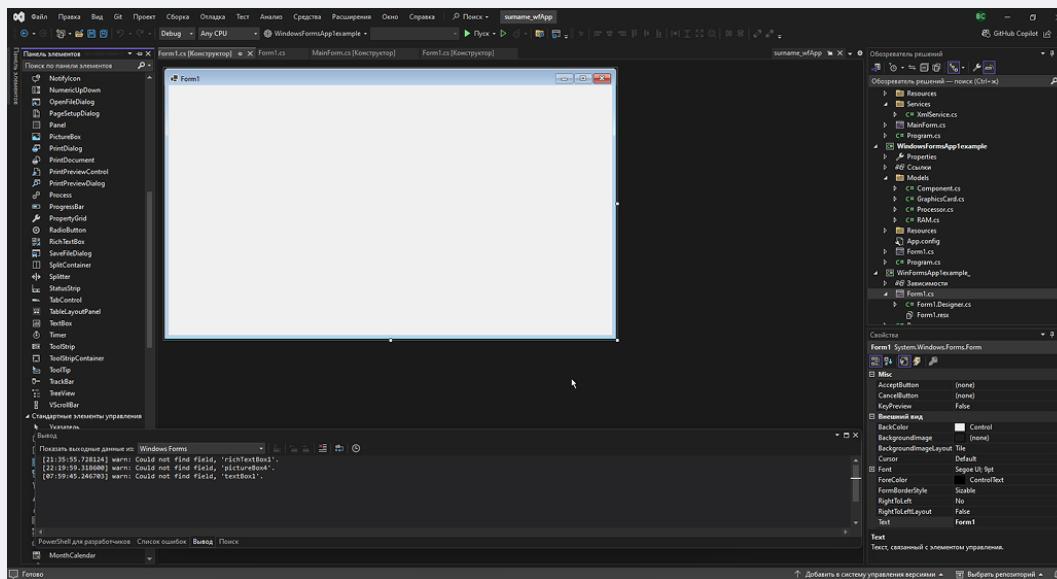


Рисунок 2.3 – Окно проекта

3 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 одностороннего 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` или `false`). Применяется в формах для подтверждения, включения настроек и т.п.
- **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 или 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, располагаемый 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.

4 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 поиска
4. PictureBox - для добавления изображения
5. Panel - для группировки элементов, создания “вкладок”
6. Label - для подписи элементов

Необходимо разместить соответствующие элементы на форме (пока что схематично). Пример формы изображен на рисунке 4.1.

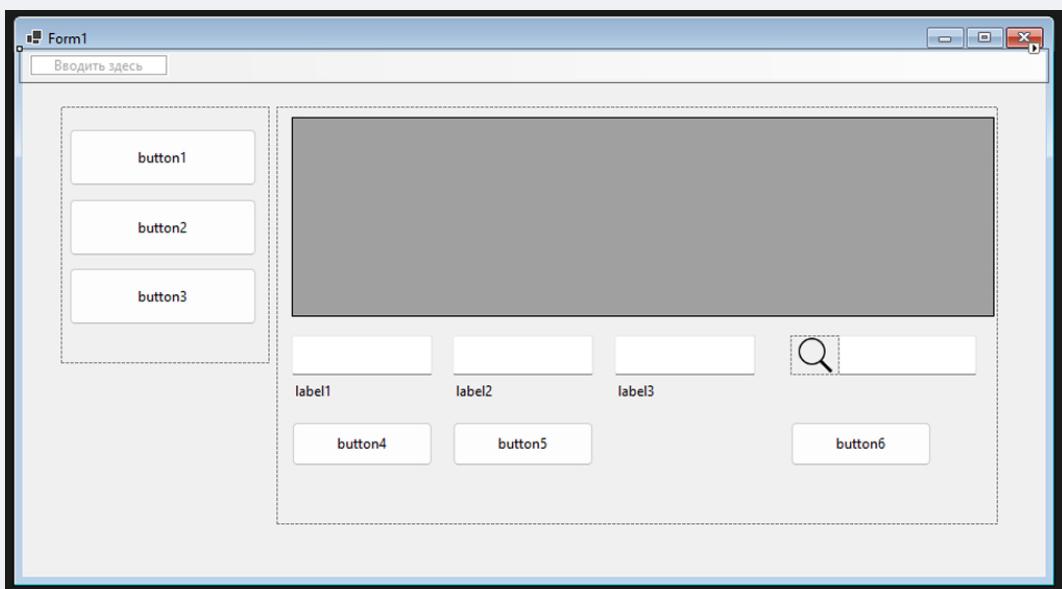


Рисунок 4.1 – Пример оформления формы

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 4.2).

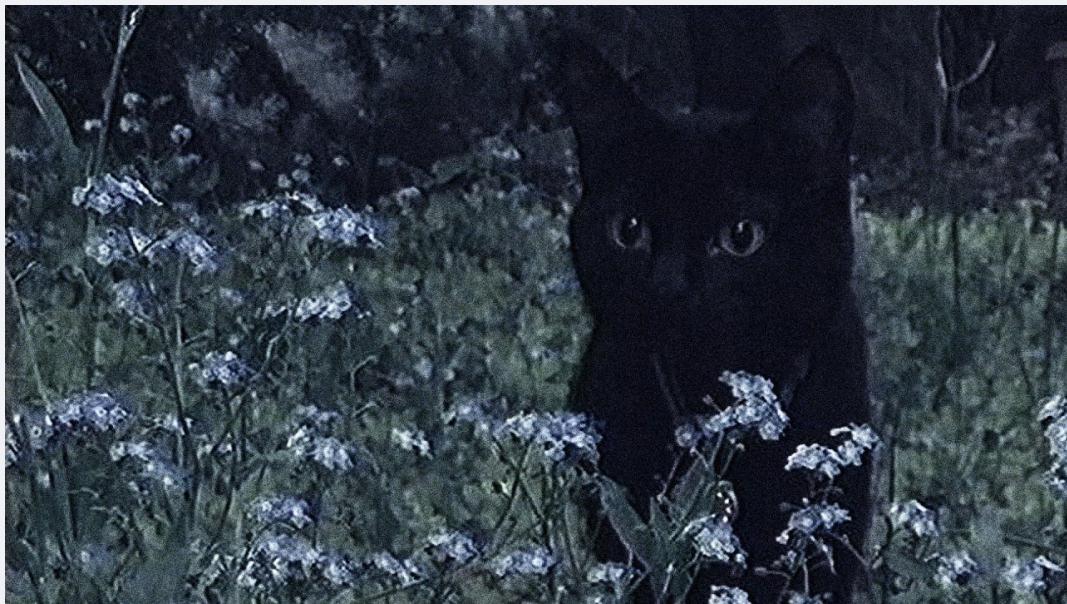


Рисунок 4.2 – Свойство Name⁴

Для того, чтобы изменить текст, отображаемый на элементе, необходимо указать новое название в свойстве “Text” в разделе “Внешний вид” (Рисунок 4.3)

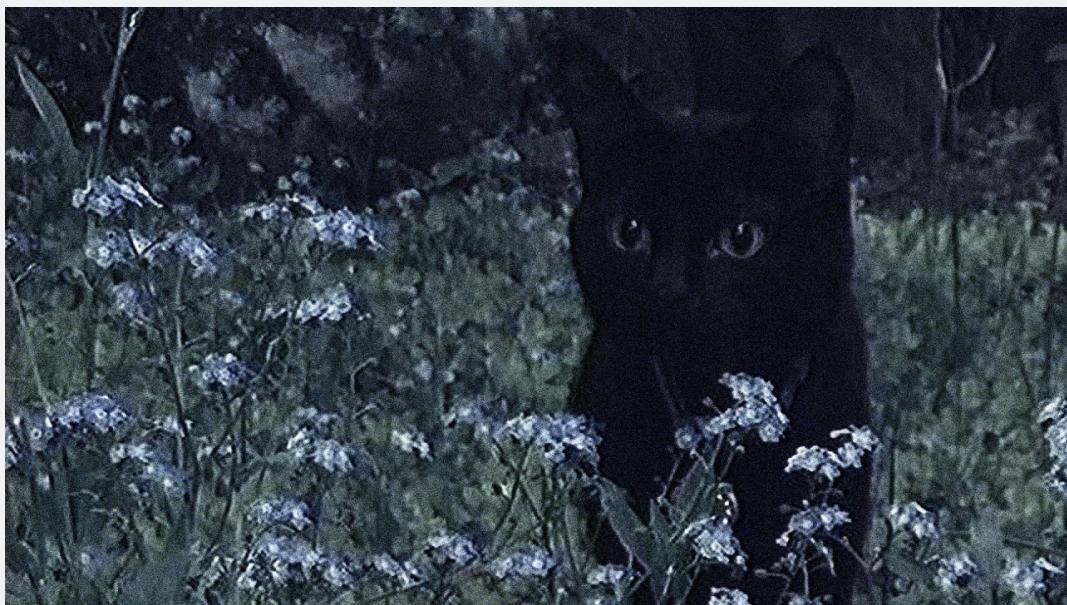


Рисунок 4.3 – Свойство Text

Также можно использовать свойство PlaceholderText, чтобы изменить текст, отображаемый на элементе. Свойство находится в разделе “Misc” (Рисунок 4.4). Данное свойство доступно при создании проекта Майкрософт

⁴Подробно про свойства можно почитать по ссылке: <https://metanit.com/sharp/windowsforms/2.2.php>

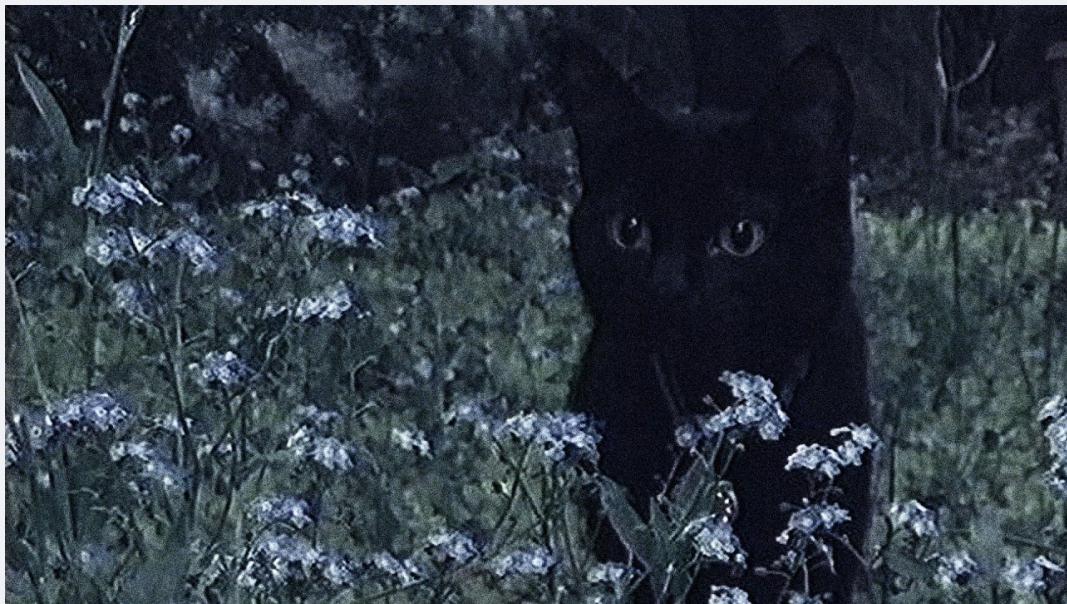


Рисунок 4.4 – Свойство Placeholder

Также необходимо настроить внешний вид элемента TextBox. Для того, чтобы изменить высоту TextBox, необходимо изменить значение параметра `Multiline` на `True` (Рисунок 4.5)

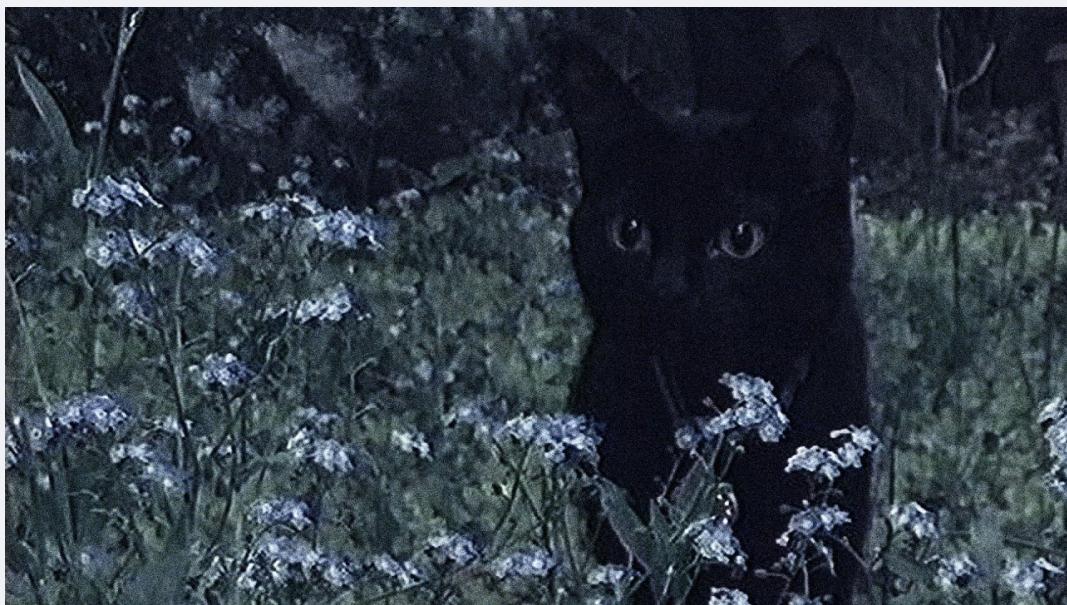


Рисунок 4.5 – Свойство Multiline

Для изменения фонового цвета используется свойство `BackColor`, а для изменения цвета текста – `ForeColor` (Рисунок 4.6)

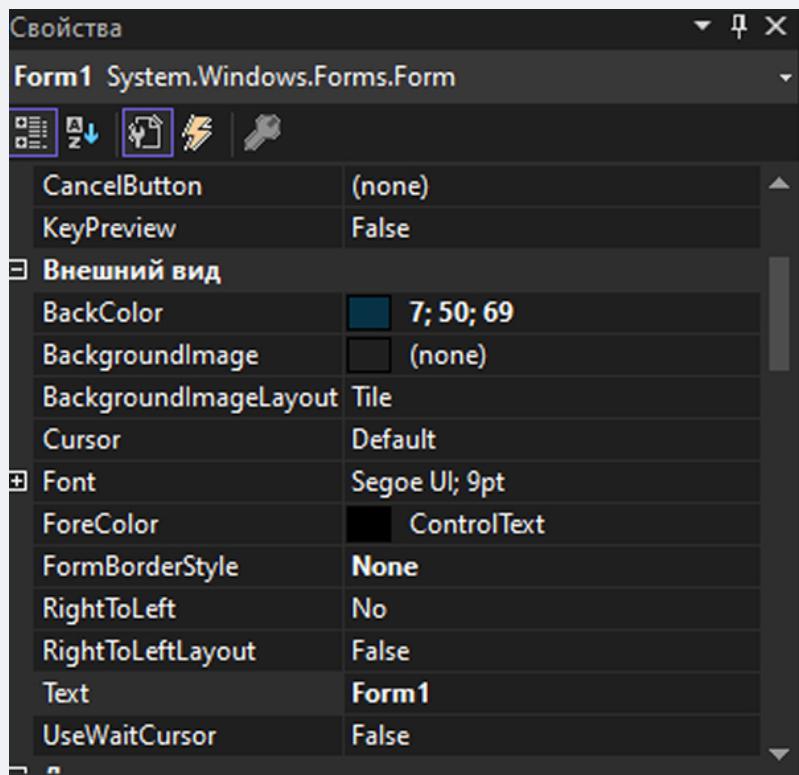


Рисунок 4.6 – Свойства для изменения цвета

Применив необходимые изменения, получим форму, изображенную на рисунке 4.7

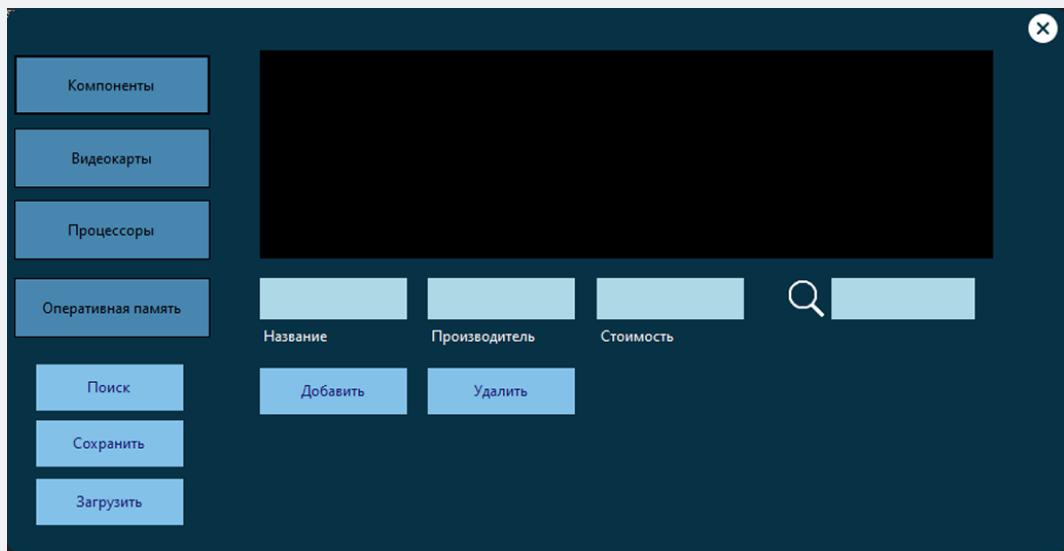


Рисунок 4.7 – Пример стилизованной формы⁵

Важно учесть расположение форм, которые в данном случае находятся друг над другом. Расположение панелей на форме указано на рисунке 4.8.

⁵Стилизация интерфейса на ваше усмотрение ☺

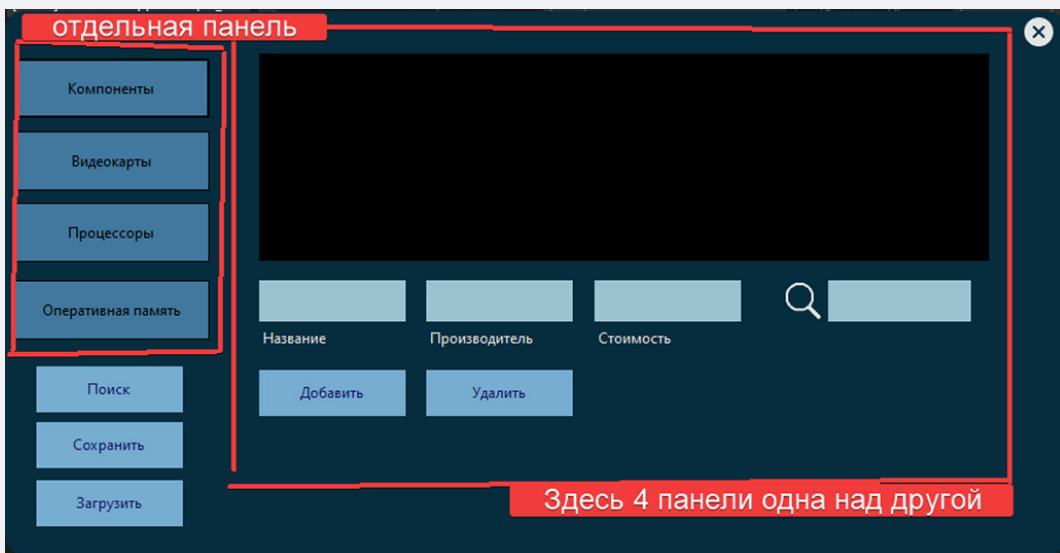
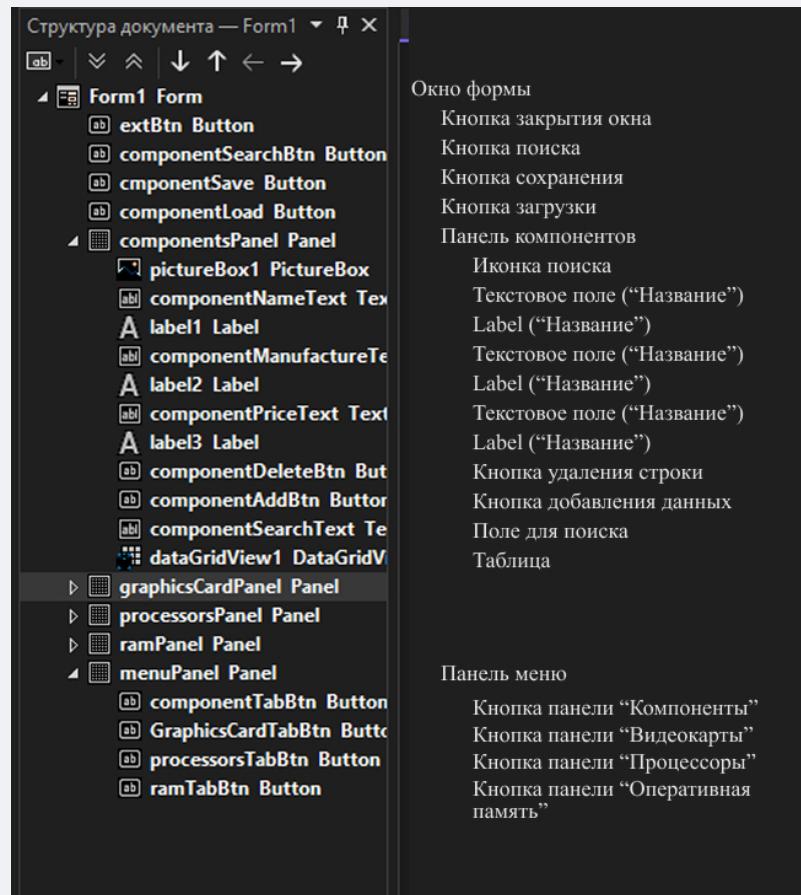


Рисунок 4.8 – Структура формы

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 (Форм.cs). Структура документа изображена на рисунке 4.9

Шаг 2. Создание формы



Окно формы

Кнопка закрытия окна

Кнопка поиска

Кнопка сохранения

Кнопка загрузки

Панель компонентов

Иконка поиска

Текстовое поле (“Название”)

Label (“Название”)

Текстовое поле (“Название”)

Label (“Название”)

Текстовое поле (“Название”)

Label (“Название”)

Кнопка удаления строки

Кнопка добавления данных

Поле для поиска

Таблица

Панель меню

Кнопка панели “Компоненты”

Кнопка панели “Видеокарты”

Кнопка панели “Процессоры”

Кнопка панели “Оперативная память”

Рисунок 4.9 – Структура формы

5 Шаг 3. Добавление классов

Далее необходимо в Решении создать папку и добавить туда классы из лабораторной 1 (Рисунок 5.1)

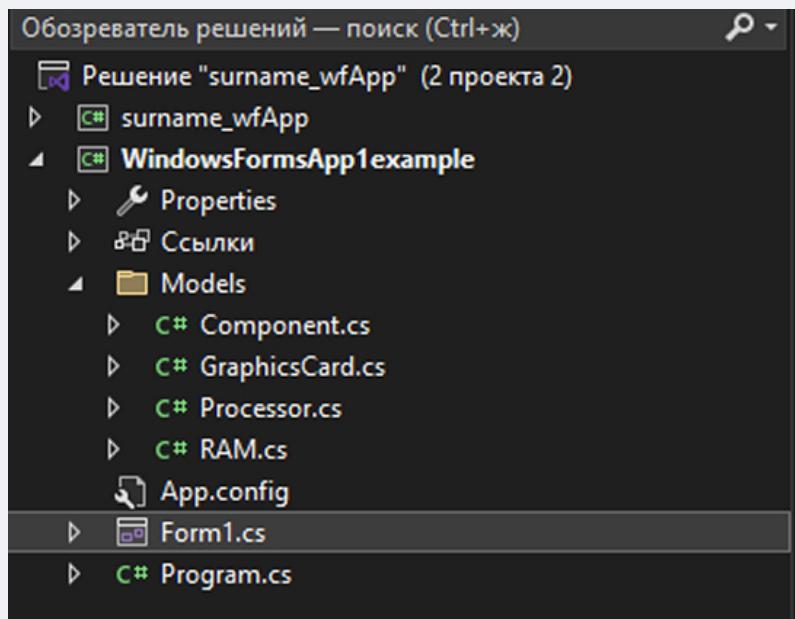


Рисунок 5.1 – Создание классов для работы с данными в форме

В каждом классе необходимо наличие 3 свойств, конструктора без параметров (нужен для сериализации) и конструктора с параметрами (Рисунок 5.1).

```
9 namespace surname_wfApp.Models
10 {
11     public class Component
12     {
13         private List<Component> components = new List<Component>();
14
15         public string Name { get; set; }
16         public string Manufacturer { get; set; }
17         public decimal Price { get; set; }
18
19         // Конструктор по умолчанию
20         public Component() { }
21
22         public Component(string name, string manufacturer, decimal price)
23         {
24             Name = name; // Устанавливаем название
25             Manufacturer = manufacturer; // Устанавливаем производителя
26             Price = price; // Устанавливаем цену
27         }
28     }
29 }
```

Листинг 5.1 – Пример структуры класса

6 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 6.1)

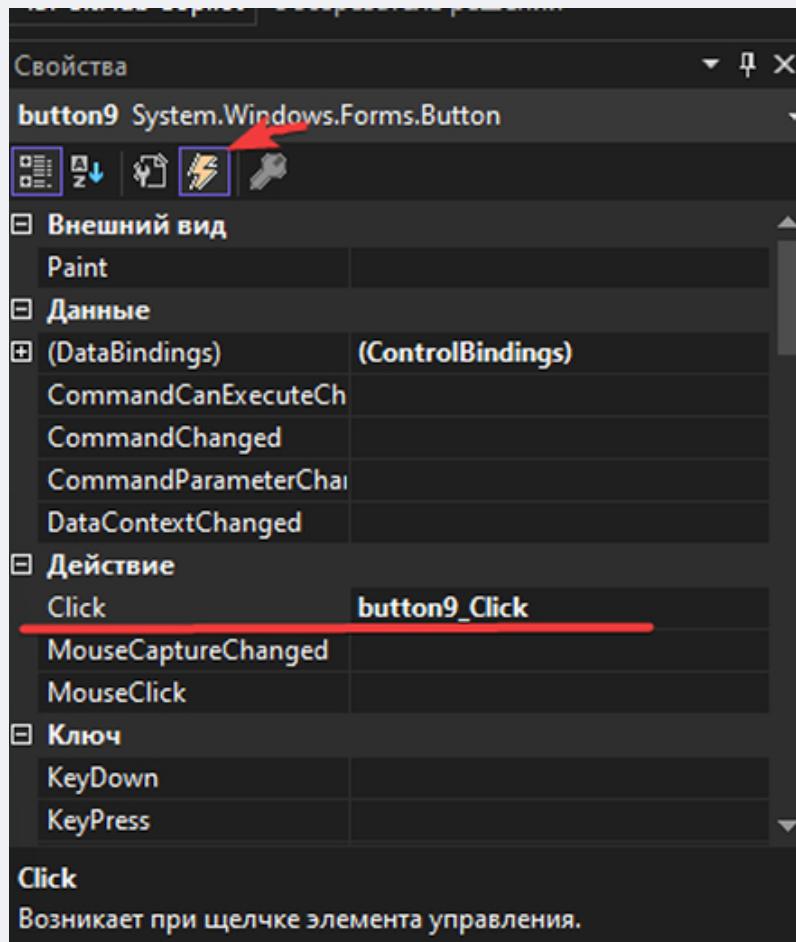


Рисунок 6.1 – Добавление событий

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

Далее необходимо добавить событие нажатия на кнопку добавления данных в таблицу. Пример метода для добавления данных в форму представлен на рисунке 6.1

Шаг 4. Реализация добавления данных в таблицу

```
162     dataGridView4.Columns["Cores"].DataPropertyName = "Cores";
163 }
164
165 // Список компонентов (например, процессоры, видеокарты и т.д.)
166 private List<Component> componentList = new List<Component>();
167
168 private void componentAddBtn_Click_1(object sender, EventArgs e)
169 {
170     dataGridView1.AutoGenerateColumns = true;
171
172     // Проверяем заполненность полей
173     if (string.IsNullOrWhiteSpace(componentNameText.Text) ||
174         string.IsNullOrWhiteSpace(componentManufactureText.Text) ||
175         string.IsNullOrWhiteSpace(componentPriceText.Text))
176     {
177         MessageBox.Show("Заполните все поля!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
178         return;
179     }
180
181     // Преобразуем строковые значения в необходимые числовые типы
182     if (!decimal.TryParse(componentPriceText.Text, out decimal price))
183     {
184         MessageBox.Show("Введите корректное значение цены!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
185         return;
186     }
187
188     // Создаем новый объект Component и добавляем его в List
189     Component newComponent = new Component(componentNameText.Text, componentManufactureText.Text, price);
190     componentList.Add(newComponent);
191
192     // Очистка текстовых полей
193     componentNameText.Clear();
194     componentManufactureText.Clear();
195     componentPriceText.Clear();
196
```

Листинг 6.1 – Метод добавления данных в форму

Также необходимо после добавления данных обновить данные в таблице (DataGridView) (Рисунок 6.2)

Шаг 4. Реализация добавления данных в таблицу

```
230     {
231         FormatPriceCell(dataGridView4, e);
232     }
233
234     private void UpdateDataGridView()
235     {
236         // Переменные для хранения активного DataGridView и списка данных
237         DataGridView activeGridView = null;
238         IEnumerable<object> activeList = null;
239
240         // Определяем, какой DataGridView и список данных использовать
241         if (componentsPanel.Visible) // Панель компонентов
242         {
243             activeGridView = dataGridView1;
244             activeList = filteredComponentList ?? componentList;
245             // Используем отфильтрованный список, если он существует
246         }
247         else if (graphicsCardPanel.Visible) // Панель видеокарт
248         {
249             activeGridView = dataGridView2;
250             activeList = filteredGraphicsCardList ?? graphicsCardList;
251         }
252         else if (processorsPanel.Visible) // Панель процессоров
253         {
254             activeGridView = dataGridView4;
255             activeList = filteredProcessorsList ?? processorsList;
256         }
257
258         // Если активный DataGridView и список определены
259         if (activeGridView != null && activeList != null)
260         {
261             // Сбрасываем привязку данных
262             activeGridView.DataSource = null;
263
264             // Присваиваем обновленный список данных DataGridView
265             activeGridView.DataSource = activeList.ToList(); // Преобразуем в список, если это не List<T>
266         }
267         // Очистить все строки, чтобы избежать дублирования данных
268         dataGridView4.DataSource = null;
269
270         // Добавляем новые строки вручную для каждого объекта в списке
271         foreach (var processor in processorsList)
272         {
273             // Добавляем строку и указываем значения для каждого столбца
```

Листинг 6.2 – Метод обновления Datagridview

Можно также добавить событие удаления элемента (строки) из таблицы (Рисунок 6.3)

Шаг 4. Реализация добавления данных в таблицу

```
342  
343  
344  
345  
346     private void componentDeleteBtn_Click(object sender, EventArgs e)  
347     {  
348         if (dataGridView1.SelectedRows.Count == 0)  
349         {  
350             MessageBox.Show("Выберите элемент для удаления!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);  
351             return;  
352         }  
353  
354         // Получаем выбранную строку из DataGridView  
355         DataGridViewRow selectedRow = dataGridView1.SelectedRows[0];  
356         // Получаем объект Component из DataSource строки  
357         Component selectedComponent = (Component)selectedRow.DataBoundItem;  
358  
359         // Удаляем объект из основного списка
```

Листинг 6.3 – Метод удаления строки из Datagridview

Для добавления данных в заранее созданные поля, необходимо связать название поля с свойствами создаваемого объекта (Рисунок 6.4)

```
138     {  
139         if (!(control is System.Windows.Forms.TextBox) && !(control is System.Windows.Forms.Button)) // Исключаем  
140             ← TextBox и Button  
141         {  
142             control.ForeColor = color;  
143         }  
144     }  
145  
146     private void Form1_Load(object sender, EventArgs e)  
147     {  
148         // Стилизация GroupBox  
149         ChangeControlsForeColor(groupBoxGC, Color.White);  
150         groupBox12.ForeColor = this.BackColor; // Цвет рамки в цвет формы  
151         groupBoxGC.ForeColor = this.BackColor; // Цвет рамки в цвет формы  
152  
153         //связывают кнопки сохранения/загрузки с DataGridView, к которому они относятся.  
154         componentSave.Tag = dataGridView1;  
155         componentLoad.Tag = dataGridView1;
```

Листинг 6.4 – Добавления данных в столбцы DataGridView без автогенерации

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 6.5)

Шаг 4. Реализация добавления данных в таблицу

```
100     private void HideAllPanels()
101     {
102         componentsPanel.Visible = false;
103         graphicsCardPanel.Visible = false;
104         processorsPanel.Visible = false;
105         ramPanel.Visible = false;
106     }
107
108     //Кнопки переключения вкладок
109     private void componentTabBtn_Click(object sender, EventArgs e)
110     {
111         HideAllPanels();
112         componentsPanel.Visible = true;
113     }
114
115     private void GraphicsCardTabBtn_Click(object sender, EventArgs e)
116     {
117         HideAllPanels();
118         graphicsCardPanel.Visible = true;
119     }
120
121     private void ramTabBtn_Click(object sender, EventArgs e)
122     {
123         HideAllPanels();
124         ramPanel.Visible = true;
125     }
126
127     private void processorsTabBtn_Click(object sender, EventArgs e)
128     {
129         HideAllPanels();
130         processorsPanel.Visible = true;
131     }
```

Листинг 6.5 – Метод скрытия панелей

7 Шаг 5. Реализация фильтрации (поиска) данных

Для поиска нужных данных в таблице использует поиск по подстроке в каждом столбце (листинг 7.1).

```

280     private void componentSearchBtn_Click(object sender, EventArgs e)
281     {
282         string query = string.Empty; // Объявляем переменную для хранения поискового запроса
283
284         // Определяем активную панель и берём текст из соответствующего поля ввода
285         if (componentsPanel.Visible) // Если активна панель компонентов
286             query = componentSearchText.Text.Trim().ToLower(); // Получаем текст из поля поиска компонентов,
287                                         // убираем пробелы и приводим к нижнему регистру
288         else if (graphicsCardPanel.Visible) // Если активна панель видеокарт
289             query = GCSearchText.Text.Trim().ToLower(); // Получаем текст из поля поиска видеокарт,
290                                         // убираем пробелы и приводим к нижнему регистру
291         else if (processorsPanel.Visible) // Если активна панель процессоров
292             query = processorSearchText.Text.Trim().ToLower(); // Получаем текст из поля поиска процессоров,
293                                         // убираем пробелы и приводим к нижнему регистру
294         else // Если ни одна панель не активна
295             return; // Завершаем выполнение метода
296
297         // Проверяем, пустой ли запрос
298         if (string.IsNullOrEmpty(query)) // Если запрос пустой
299         {
300             filteredComponentList = null; // Сбрасываем фильтр для компонентов
301             filteredGraphicsCardList = null; // Сбрасываем фильтр для видеокарт
302             filteredProcessorsList = null; // Сбрасываем фильтр для процессоров
303             UpdateDataGridView(); // Обновляем таблицу для отображения полного списка
304             return; // Завершаем выполнение метода
305         }
306
307         // Выполняем фильтрацию в зависимости от активной панели
308         if (componentsPanel.Visible) // Если активна панель компонентов
309         {
310             filteredComponentList = componentList.Where(comp => // Фильтруем список компонентов
311                 comp.Name.ToLower().Contains(query) || // Поиск по имени (без учёта регистра)
312                 comp.Manufacturer.ToLower().Contains(query) || // Поиск по производителю (без учёта регистра)
313                 comp.Price.ToString().Contains(query) // Поиск по цене (как строке)
314             ).ToList(); // Преобразуем результат в список
315         }
316         else if (graphicsCardPanel.Visible) // Если активна панель видеокарт
317         {
318             filteredGraphicsCardList = graphicsCardList.Where(gc => // Фильтруем список видеокарт
319                 gc.Name.ToLower().Contains(query) || // Поиск по имени (без учёта регистра)
320                 gc.Manufacturer.ToLower().Contains(query) || // Поиск по производителю (без учёта регистра)
321                 gc.Price.ToString().Contains(query) || // Поиск по цене (как строке)
322                 gc.MemorySize.ToString().Contains(query) || // Поиск по объёму памяти (как строке)
323                 gc.Type.ToLower().Contains(query) || // Поиск по типу памяти (без учёта регистра)
324                 gc.BusWidth.ToString().Contains(query) // Поиск по ширине шины (как строке)
325             ).ToList(); // Преобразуем результат в список
326         }
327         else if (processorsPanel.Visible) // Если активна панель процессоров
328         {
329             filteredProcessorsList = processorsList.Where(proc => // Фильтруем список процессоров
330                 proc.Name.ToLower().Contains(query) || // Поиск по имени (без учёта регистра)
331                 proc.Manufacturer.ToLower().Contains(query) || // Поиск по производителю (без учёта регистра)
332                 proc.Price.ToString().Contains(query) || // Поиск по цене (как строке)
333                 proc.Frequency.ToString().Contains(query) || // Поиск по частоте (как строке)
334                 proc.Cores.ToString().Contains(query) // Поиск по количеству ядер (как строке)
335             ).ToList(); // Преобразуем результат в список
336         }
337
338         UpdateDataGridView(); // Обновляем таблицу для отображения отфильтрованных данных
339     }
340 }
```

Листинг 7.1 – Метод фильтрации (поиска) данных в таблице

8 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

- **Binary** (Бинарная) — сериализация в двоичный формат (быстро, но неудобно для чтения)
- **XML** (Текстовая) — сохраняет объект в формате XML (читаемый формат, но объемный)
- **JSON** (Современный стандарт) — сериализация в JSON (удобно для веба, поддерживается большинством языков)
- **Custom** (Кастомная) — своя реализация, например, в CSV⁶.

В данном случае будет рассмотрена XML-сериализация. В C# для работы с XML используется класс `XmlSerializer` из пространства имен `System.Xml.Serialization`.

Пример метода сериализации с использованием `List` приведен на рисунке 8

```

531     private void SaveDataGridView(DataGridView dgv)
532     {
533         SaveFileDialog saveFileDialog = new SaveFileDialog(); // Создаём объект SaveFileDialog для выбора пользователем
534         ← пути и имени файла
535         saveFileDialog.Filter = "XML Files (*.xml)|*.xml|All Files (*.*)|*.*"; // Устанавливаем фильтр для отображения
536         ← XML-файлов по умолчанию и опции "все
537         ← файлы"
538
539         if (saveFileDialog.ShowDialog() == DialogResult.OK) // Открываем диалог сохранения и проверяем, выбрал ли
540             ← пользователь файл (нажал "OK")
541         {
542             string filePath = saveFileDialog.FileName; // Получаем полный путь к файлу, выбранному пользователем
543             ← (например, "C:\data.xml")
544
545             if (dgv == null) // Проверяем, был ли передан действительный DataGridView (не null)
546             {
547                 MessageBox.Show("Нет доступных DataGridView для сохранения."); // Если DataGridView не передан,
548                 ← показываем сообщение об ошибке
549                 return; // Завершаем выполнение метода
550             }
551
552             // Выбираем, какой список сериализовать в зависимости от переданного DataGridView
553             if (dgv == dataGridView1) // Если передан DataGridView для компонентов (dataGridView1)
554             {
555                 if (componentList.Count == 0) // Проверяем, содержит ли список componentList данные
556                 {
557                     MessageBox.Show("Нет данных для сохранения."); // Если список пуст, показываем сообщение
558                 }
559             }
560         }
561     }
562 
```

⁶Это простой текстовый формат для хранения табличных данных, где каждая строка файла представляет одну строку таблицы. Значения в каждой строке разделяются определенным символом, чаще всего запятой.

```

553         return; // Завершаем выполнение метода
554     }
555
556     XmlSerializer serializer = new XmlSerializer(typeof(List<Component>)); // Создаём XmlSerializer для
557     // сериализации
558     using (TextWriter writer = new StreamWriter(filePath)) // Создаём TextWriter (StreamWriter) для записи
559     // данных в файл
560     {
561         serializer.Serialize(writer, componentList); // Сериализуем componentList в XML и записываем в файл
562     } // Автоматически закрываем writer благодаря using
563     MessageBox.Show("Данные успешно сохранены."); // Сообщаем пользователю об успешном сохранении
564 }
565 else if (dgv == dataGridView2) // Если передан DataGridView для видеокарт (dataGridView2)
566 {
567     if (graphicsCardList.Count == 0) // Проверяем, содержит ли список graphicsCardList данные
568     {
569         MessageBox.Show("Нет данных для сохранения."); // Если список пуст, показываем сообщение
570         return; // Завершаем выполнение
571     }
572     Debug.WriteLine($"Graphics cards count: {graphicsCardList.Count}");
573
574     XmlSerializer serializer = new XmlSerializer(typeof(List<GraphicsCard>)); // Создаём XmlSerializer для
575     // сериализации
576                                         // списка List<GraphicsCard> в
577                                         // XML
578     using (TextWriter writer = new StreamWriter(filePath)) // Создаём TextWriter для записи данных в файл
579     {
580         serializer.Serialize(writer, graphicsCardList); // Сериализуем graphicsCardList в XML и записываем в
581         // файл
582     } // Закрываем writer
583     MessageBox.Show("Данные успешно сохранены."); // Сообщаем об успешном сохранении
584 }
585 else if (dgv == dataGridView4) // Если передан DataGridView для процессоров (dataGridView4)
586 {
587     if (processorsList.Count == 0) // Проверяем, содержит ли список processorsList данные
588     {
589         MessageBox.Show("Нет данных для сохранения."); // Если список пуст, показываем сообщение
590         return; // Завершаем выполнение
591     }
592
593     XmlSerializer serializer = new XmlSerializer(typeof(List<Processor>)); // Создаём XmlSerializer для
594     // сериализации
595                                         // списка List<Processor> в XML
596     using (TextWriter writer = new StreamWriter(filePath)) // Создаём TextWriter для записи данных в файл
597     {
598         serializer.Serialize(writer, processorsList); // Сериализуем processorsList в XML и записываем в файл
599     } // Закрываем writer
600     MessageBox.Show("Данные успешно сохранены."); // Сообщаем об успешном сохранении
601 }
602 }
603 }

```

Объяснение ключевых моментов

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



- **using (TextWriter ...):** Обеспечивает корректное закрытие файла после записи, даже если произойдёт ошибка
- **Проверка на пустой список:** Перед сериализацией проверяется Count, чтобы не создавать пустой XML-файл
- **Условные ветви (if-else):** Код определяет, какой список сериализовать, основываясь на активном DataGridView. Это позволяет сохранять данные только для текущей вкладки.

Также можно использовать сериализацию данных с помощью DataTable, не используя списки List (Рисунок 8.1)

```

531     private void SaveDataGridView(DataGridView dgv)
532     {
533         SaveFileDialog saveFileDialog = new SaveFileDialog(); // Создаём объект SaveFileDialog для выбора пользователем
534         ← пути и имени файла
535         saveFileDialog.Filter = "XML Files (*.xml)|*.xml|All Files (*.*)|*.*"; // Устанавливаем фильтр для отображения
536         ← XML-файлов по умолчанию и опции "все
537         ← файлы"
538
539         if (saveFileDialog.ShowDialog() == DialogResult.OK) // Открываем диалог сохранения и проверяем, выбрал ли
540             ← пользователь файл (нажал "OK")
541         {
542             string filePath = saveFileDialog.FileName; // Получаем полный путь к файлу, выбранному пользователем
543             ← (например, "C:\data.xml")
544
545             if (dgv == null) // Проверяем, был ли передан действительный DataGridView (не null)
546             {
547                 MessageBox.Show("Нет доступных DataGridView для сохранения."); // Если DataGridView не передан,
548                 ← показываем сообщение об ошибке
549                 return; // Завершаем выполнение метода
550             }
551
552             // Выбираем, какой список сериализовать в зависимости от переданного DataGridView
553             if (dgv == dataGridView1) // Если передан DataGridView для компонентов (dataGridView1)
554             {
555                 if (componentList.Count == 0) // Проверяем, содержит ли список componentList данные

```

Листинг 8.1 – Пример сериализации данных с помощью DataTable

Шаг 6. Сериализация, десериализация классов. Сохранение в XML

```
467 private void LoadDataGridView(DataGridView dgv) // Метод для загрузки данных из XML-файла в список,
468 // привязанный к переданному DataGridView
469 {
470     OpenFileDialog openFileDialog = new OpenFileDialog(); // Создаём диалог для выбора файла пользователем
471     openFileDialog.Filter = "XML Files (*.xml)|*.xml|All Files (*.*)|*.*"; // Устанавливаем фильтр:
472 // XML-файлы по умолчанию, с опцией "все
473 // файлы"
474
475     if (openFileDialog.ShowDialog() == DialogResult.OK) // Открываем диалог и проверяем, выбрал ли пользователь файл
476     {
477         string filePath = openFileDialog.FileName; // Получаем путь к выбранному файлу (например, "C:\data.xml")
478
479         if (dgv == null) // Проверяем, передан ли действительный DataGridView
480         {
481             MessageBox.Show("Передан некорректный DataGridView для загрузки данных."); // Если dgv null, показываем
482 // ошибку
483             return; // Завершаем выполнение
484         }
485
486         // Загружаем данные в соответствующий список в зависимости от переданного DataGridView
487         if (dgv == dataGridView1) // Если передан DataGridView для компонентов
488         {
489             XmlSerializer serializer = new XmlSerializer(typeof(List<Component>)); // Создаём сериализатор для
490 // List<Component>
491             using (TextReader reader = new StreamReader(filePath)) // Открываем поток чтения из файла
492             {
493                 componentList = (List<Component>)serializer.Deserialize(reader); // Десериализуем XML в componentList
494             }
495             filteredComponentList = null; // Сбрасываем фильтр для отображения полного списка
496         }
497         else if (dgv == dataGridView2) // Если передан DataGridView для видеокарт
498         {
499             XmlSerializer serializer = new XmlSerializer(typeof(List<GraphicsCard>)); // Создаём сериализатор для
500 // List<GraphicsCard>
501             using (TextReader reader = new StreamReader(filePath)) // Открываем поток чтения
502             {
503                 graphicsCardList = (List<GraphicsCard>)serializer.Deserialize(reader); // Десериализуем XML в
504 // graphicsCardList
505             }
506             filteredGraphicsCardList = null; // Сбрасываем фильтр
507         }
508         else if (dgv == dataGridView4) // Если передан DataGridView для процессоров
509         {
510             XmlSerializer serializer = new XmlSerializer(typeof(List<Processor>)); // Создаём сериализатор для
511 // List<Processor>
512             using (TextReader reader = new StreamReader(filePath)) // Открываем поток чтения
513             {
514                 processorsList = (List<Processor>)serializer.Deserialize(reader); // Десериализуем XML в
515 // processorsList
516             }
517             filteredProcessorsList = null; // Сбрасываем фильтр
518         }
519         else // Если передан неизвестный DataGridView
520         {
521             MessageBox.Show("Неизвестный DataGridView для загрузки данных."); // Сообщаем об ошибке
522             return; // Завершаем выполнение
523         }
524
525         UpdateDataGridView(); // Обновляем активный DataGridView с новыми данными
526         MessageBox.Show("Данные успешно загружены."); // Сообщаем об успехе
527     }
528 }
```

Листинг 8.2 – Пример метода десериализации⁷

⁷Десериализация — это процесс обратного преобразования, то есть восстановления объекта из сохраненного состояния



Разбор метода десериализации

```
467     private void LoadDataGridView(DataGridView dgv) // Метод для загрузки данных из XML-файла в список,
468                                     // привязанный к переданному DataGridView
```

Листинг 8.3 – 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 8.3)

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

Листинг 8.4 – 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 8.4)

```
474     if (openFileDialog.ShowDialog() == DialogResult.OK) // Открываем диалог и проверяем, выбрал ли пользователь файл
475     {
476         string filePath = openFileDialog.FileName; // Получаем путь к выбранному файлу (например, "C:\data.xml")
477
478         if (dgv == null) // Проверяем, передан ли действительный DataGridView
479         {
480             MessageBox.Show("Передан некорректный DataGridView для загрузки данных."); // Если dgv null, показываем
                                         // ошибку
481             return; // Завершаем выполнение
482     }
```

Листинг 8.5 – Выбор файла и проверка на корректную таблицу

Метод `ShowDialog()` (Рисунок 8.5) открывает диалоговое окно выбора файла. Если пользователь выбрал файл и нажал ”Открыть” возвращается `DialogResult.OK`. Если пользователь нажал ”Отмена” метод завершится, ничего не делая.

```

1 if (dgv == null)
2 {
3     MessageBox.Show("Передан некорректный DataGridView для загрузки данных.");
4     return;
5 }

```

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.

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.

```

485         if (dgv == dataGridView1) // Если передан DataGridView для компонентов
486         {
487             XmlSerializer serializer = new XmlSerializer(typeof(List<Component>)); // Создаём сериализатор для
488             // List<Component>
489             using (TextReader reader = new StreamReader(filePath)) // Открываем поток чтения из файла
490             {
491                 componentList = (List<Component>)serializer.Deserialize(reader); // Десериализуем XML в componentList
492             }
493             filteredComponentList = null; // Сбрасываем фильтр для отображения полного списка

```

Листинг 8.6 – Сериализация XML

Далее рассмотрим объект `XmlSerializer`, позволяющий сохранять данные из таблицы в формате XML (Рисунок 8.6)

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

Создаётся объект `XmlSerializer` для типа `List<Component>`. `XmlSerializer` знает, как преобразовать XML в список объектов `Component`

указывает, что ожидается XML-файл, содержащий список объектов `Component`

`StreamReader` открывает файл по пути `filePath` для чтения текста Конструкция `using` гарантирует, что поток (`StreamReader`) будет закрыт после использования, даже если произойдёт исключение

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`

Сбрасывается фильтр, чтобы после загрузки отображался полный список компонентов, а не отфильтрованный.

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)

Рисунок 6.8 Связь событий и кнопок с помощью Tag

9 Дополнительная информация ?

1. Обязательно ли указывать атрибут Serializable?

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

2. 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)

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:

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

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

Если DataGridView использует DataTable как источник данных:

```
1 // Получаем доступ к DataTable
2 DataTable dt = (DataTable)dataGridView1.DataSource; // Изменяем данные в нужной строке и столбце
3 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:

```
1 // Предположим, нас есть BindingList<MyObject> bindingList[0].SomeProperty =
2 "Новое значение"; dataGridView1.Refresh();
3 // 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.

10 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`

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

Рисунок 6.10 Функции реализации перемещения формы

Обработчики нажатия мыши приведены на рисунке 6.11

Рисунок 6.11 Обработчики движения мыши

Также в примере применяется изменение цвета элементов `label`, размещенных внутри `groupbox`

Рисунок 6.12

Рисунок 6.12 Обработчики движения мыши

11 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)

Рисунок 6.13 Форматирование столбца `Price`

Рисунок 6.14 Метод форматирования строки с стоимостью А затем вызывать этот метод у каждой таблицы (свойство `CellFormatting`) Рисунок 6.15

Рисунок 6.15 Вызов метода форматирования

12 Интерфейс приложения 60

Далее представлены примеры интерфейса приложения. Добавление данных в таблицу Components представлено на рисунке 6.16

Рисунок 6.16 Добавление данных в таблицу компонентов Удаление осуществляется нажатием на соответствующую строку и затем нажатием кнопки “Удалить” (Рисунок 6.17)

Рисунок 6.17 Удаление данных из таблицы компонентов 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 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)

Рисунок 6.19 Добавление данных в таблицу компонентов Также реализовано добавление данных в таблицы других классов (Рисунок 6.20)

Рисунок 6.20 Добавление данных в таблицу видеокарт

13 Вопросы для защиты

1. Вопрос
2. Вопрос
3. Вопрос
4. Вопрос
5. Вопрос
6. Вопрос
7. Вопрос
8. Вопрос
9. Вопрос
10. Вопрос