ЛАБОРАТОРНАЯ РАБОТА №1.
БИБЛИОТЕКИ

СОДЕРЖАНИЕ

Цель

Научиться создавать и подключать библиотеки классов. Ознакомиться с концепциями DAL, IoC.

Задание

1. Создать новый проект с репозиторием. Создать ветку от главной ветки (для 1 лабораторной).
2. *Требуется сделать приложение по учету изготовления изделий на заказ. Необходимо реализовать следующие требования:*
    a. *Ввод списка компонент, используемых при изготовлении изделий. Каждый компонент имеет уникальное имя, отличающее его от других компонент и цену за единицу компонента.*
    б. *Ввод списка производимых изделий. Для каждого изделия предусмотреть возможность указания компонент, из которых оно изготавливается и в каком количестве каждый компонент требуется при изготовлении изделия. Каждое изделие имеет стоимость, а также уникальное имя, отличающее его от других изделий. Стоимость изделия рассчитывается из суммы компонент, которые используются при его создании, и добавления определенного процента к этой сумме.*
    в. *Создание заказов. В заказ добавляется только одно изделие в любом количестве.*
    г. *Необходимо фиксировать дату создания заказа и дату выполнения.*
    д. *Заказ должен проходить ряд стадий: создание, изготовление, готов к выдачи, выдача заказа.*

Приложение должно быть оформлено в виде desktop-приложения. Требуется реализовать способ хранения данных в оперативной

памяти (все хранится, пока приложение работает, после закрытия приложения данные теряются).

3. Вылить полученный результат в созданную ветку. Убедится, что там нет лишних файлов (типа .exe или .bin). Создать pull request.

Проектирование

На основе задания, выделим 3 сущности:

- компонент;
- изделие;
- заказ.

По связям:

- компонент используется в изделии, при чем может фигурировать несколько компонент в изделии;
- изделие используется в заказе в единственном экземпляре;

Так как названия компонента или изделия могут меняться, для корректности связей между сущностями, у каждой из них объявим элемент «идентификатор», через которые и будем связывать сущности. Например, в заказе будет фигурировать не название изготавливаемого изделия, а его идентификатор. Идентификатор задается при создании записи сущности и более не меняется. И в рамках одной сущности не может быть двух записей с одинаковым идентификатором, т.е. он должен быть уникальным.

По элементам сущностей (помимо идентификатора):

- для компонента требуется хранить его название и цену;
- для изделия требуется хранить название, стоимость и коллекцию компонент (какой компонент (идентификатор) и в каком количестве);
- для заказа требуется хранить изготавливаемое изделие (идентификатор), в каком количестве изготавливается, сумма заказа (так как цена изделия в будущем может поменять необходимо хранить либо сумму по заказу, либо цену изделия в

момент создания заказа, чтобы всегда была возможность правильно рассчитать сумму), дата создания заказа, дата выполнения заказа и статус заказа.

По статусу заказа лучше всего задать фиксированные варианты статуса (перечисление): принят, выполняется, готов, выдан.

В качестве архитектуры проекта выберем многоуровневую архитектуру с применением концепции Data Access Level. Данный подход позволит легко подменять способы хранения данных, заменять способ представления (интерфейс пользователя) или бизнес-логику и расширять проект в будущем на других платформах, например, в виде web-приложения.

Выделим 5 слоев в проекте. Два из них будут базовыми и использоваться в других слоях, это:

- слой с описанием моделей приложения (для улучшения синхронизации полей сущностей, в случае их расширения);
- слой с описанием возможностей бизнес-логики и манипуляции с данными, через которые будут взаимодействовать слои верхнего уровня, без лишних связей между собой.

К слоям верхнего уровня отнесем:

- слой с бизнес-логикой приложения;
- слой с различными способами хранения данных;
- слой с представлением пользователя (пользовательский интерфейс, desktop-приложение).

Рассмотрим детальнее каждый из слоев:

- Слоя с моделями. Данный слой должен содержать описание моделей сущностей, а по сути указание, какие поля/свойства должен содержать в себе каждый класс, связанный с сущностями. Описание можно задать или через класс, или через интерфейс (во втором случае описание будет задавать только через свойства). Если выбрать первый вариант, то для всех классов-наследников

встанет ограничение, что более ни от каких классов они унаследоваться не смогут, что не очень удобно, так что выберем второй вариант определения моделей сущностей, через интерфейсы. Также тут же объявляется перечисление со статусами заказа (так как это одно из свойств для сущности «Заказ»).

- Слой с описанием возможностей бизнес-логики и манипуляции с данными. В данном слое будут объявлены «договоренности» (контракты) о способах обработки данных пользователя (что он вводит, что для него выводить) и о способах работы с хранилищем (как сохранять и получать данные из хранилища). Контракты объявляются в виде интерфейсов. Будет 2 группы интерфейсов: для описания работы бизнес-логики и для описания работы с хранилищем. Также, для обмена данными между слоями заведем 3 группы классов-прослоек: для передачи данных в методы для сохранения, для передачи данных в методы для поиска и для возврата данных из методов. Можно было бы разбить еще классы-прослойки на группы обмена данными для интерфейсов бизнес-логики и интерфейсов хранилищ, но так как данные там будут одинаковые, то смысла в этом нет. Итого в слое контрактов будет 5 групп: классы-прослойки для передачи данных в методы для сохранения, для передачи данных в методы для поиска, классы-прослойки для возврата данных из методов, интерфейсы для описания работы бизнес-логики и интерфейсы для описания работы с хранилищем.

  Методы интерфейсов бизнес-логики выделим на основе того, каких целей хотим достигнуть, а именно: получение списка, получение отдельной записи, создание записи, изменение записи и удаление записи. Для сущности «Заказ» будут особые методы. Для него требуются методы получения списка, создания заказа и

набор методов для смены статуса заказа (один метод нельзя, так как в логике некоторых методов есть нюансы).

Для интерфейсов работы с хранилищами выделим методы получение полного списка, фильтрованного списка, одного элемента, добавление элемента, редактирование элемента и удаление элемента.

ВАЖНО! Данный набор методов интерфейсов не является каким-то каноном, он определяется исходя из особенностей проекта, виденья разработчика или особенностей языка разработки.

- Слой бизнес-логики. В данном слое будет создана реализация интерфейсов бизнес-логики, описанных в слое контрактов. При реализации активно будут использоваться методы, объявленные в интерфейсах хранилища.
- Слой с различными способами хранения данных. На данном этапе разработки слой будет представлен одним вариантом реализации способа хранения данных в оперативной памяти. Для реализации потребуется объявить классы-сущности на основе моделей-интерфейсов, объявленных в слое моделей. Также потребуется класс, в котором будут хранится списки от классов-моделей. Объект от этого класса будет использоваться в реализациях интерфейсов хранилища из слоя контрактов. Чтобы в каждой реализации был один и тот же набор списков, то к классу со списками следует применить паттерн Singleton, чтобы на все время программы создавался только один экземпляр этого класса. И, как было сказано выше в этом же слое будет реализация интерфейсов хранилища.
- Слой с представлением пользователя. Настольное приложение с набором форм. Будет главная форма с выводом списка заказов. На этой же форме будет меню для работы с другими сущностями

(«Компоненты», «Изделия») и кнопки для работы с заказами. Для сущности «Компонент» потребуется форма для вывода списка и форма для создания/редактирования отдельного «компонента». Для «Изделия» потребуется формы для вывода списка, для создания/редактирования отдельного «изделия» и для создания/редактирования компонента изделия. Дополнительно форма для создания заказа. В логиках форм будут активно использоваться методы, объявленные в интерфейсах бизнес-логики.

Таким образом получается следующая архитектура приложения (рисунок 1.1). Слой моделей будет взаимодействовать со слоями контрактов и способа хранения. Слой контрактов со слоями бизнес-логики, способа хранения и представления пользователя.

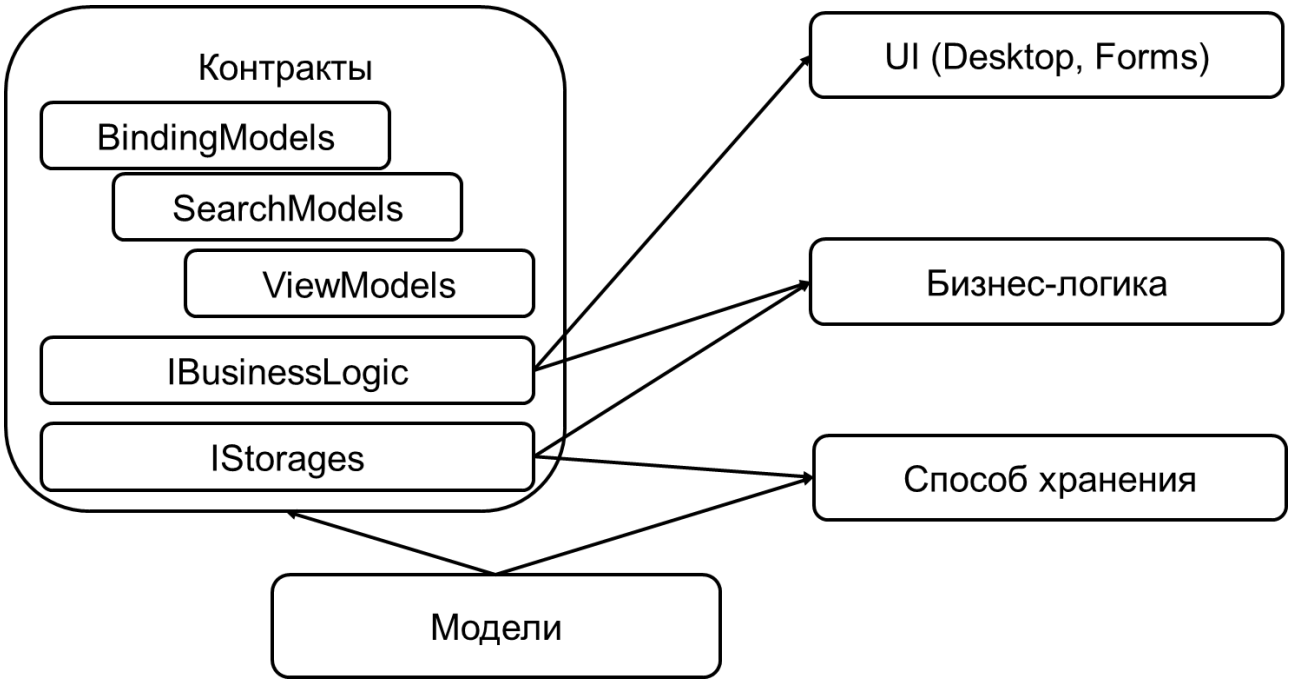

Рисунок 1.1 – Взаимосвязи слоев проекта

Технически, будет еще одна связь между слоем представления пользователя и слоями бизнес-логики и хранения данными, так как потребуется определять конкретные реализации интерфейсов, объявленных в слое контрактов и это будет проще сделать в слое представления пользователя.

Реализация

Добавим в созданный на 0 этапе решение новый проект с моделями. Тип проекта будет – библиотека классов. Выберем тип библиотеки .Net Core 6.0 (рисунок 1.2). Назовем проект «**AbstractShopDataModels**».

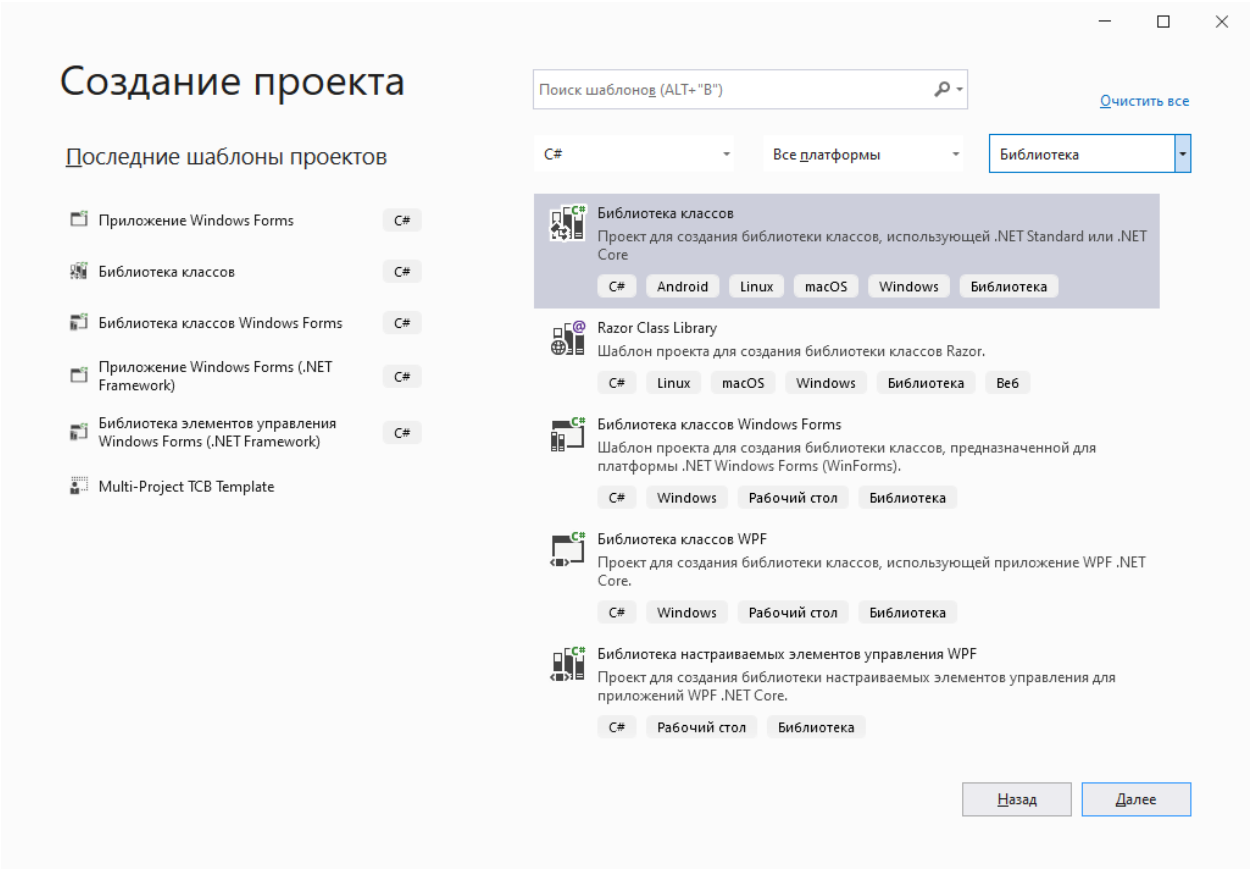

Рисунок 1.2 – Выбор типа создаваемого приложения

Нам потребуется одно перечисление (листинг 1.1) и 3 интерфейса, по одной на каждую сущность. Так как в каждом перечислении будет фигурировать идентификатор, то сделаем отдельный интерфейс, в котором его объявим, и все 3 оставшихся унаследуем от него (листинг 1.2-1.5). У свойств интерфейсов объявим только get-метод, чтобы в нужных местах закрыть возможность несанкционированного изменения свойств.

```
namespace AbstractShopDataModels.Enums
{
    public enum OrderStatus
    {
        Неизвестен = -1,

        Принят = 0,

        Выполняется = 1,
```

```
        Готов = 2,

        Выдан = 3
    }
}
```

Листинг 1.1 – Перечисление OrderStatus

```
namespace AbstractShopDataModels
{
    public interface IId
    {
        int Id { get; }
    }
}
```

Листинг 1.2 – Интерфейс IId

```
namespace AbstractShopDataModels.Models
{
    public interface IComponentModel : IId
    {
        string ComponentName { get; }

        double Cost { get; }
    }
}
```

Листинг 1.3 – Интерфейс IComponentModel

```
namespace AbstractShopDataModels.Models
{
    public interface IProductModel : IId
    {
        string ProductName { get; }

        double Price { get; }

        Dictionary<int, (IComponentModel, int)> ProductComponents { get; }
    }
}
```

Листинг 1.4 – Интерфейс IProductModel

```
using AbstractShopDataModels.Enums;

namespace AbstractShopDataModels.Models
{
    public interface IOrderModel : IId
    {
        int ProductId { get; }

        int Count { get; }

        double Sum { get; }

        OrderStatus Status { get; }

        DateTime DateCreate { get; }

        DateTime? DateImplement { get; }
    }
}
```

Листинг 1.5 – Интерфейс IOrderModel

Значение «DateTime? DateImplement» говорит о том, что при создании записи о заказе не требуется указывать дату выполнения заказа, она может хранить в себе значения null. По логике это означает, что в момент оформления заказа мы не знаем дату, когда он будет выполнен.

Далее создадим проект с контрактами для классов BindingModel, SearchModel, ViewModel, и интерфейсов бизнес-логики и хранилища. Тип проекта будет – библиотека классов, платформа .Net Core 6.0. Назовем проект «**AbstractShopContracts**». Сразу добавим ссылку на проект с моделями. Для этого откроем вкладку «Обозреватель решения», найдем созданный проект, кликнем правой кнопкой мыши по пункту «Ссылки» и выберем пункт «Добавить ссылку…». Далее, перейдем на вкладку «Проекты» и поставить галочку напротив проекта «**AbstractShopDataModels**».

В проекте сделаем 5 папок. Одна из папок предназначена для интерфейсов с описанием методов бизнес-логики. Три папки для уменьшения связанности классов (принцип SOLID) вводится еще три группы классов. BindingModels – для классов, в которых будут передаваться данные от интерфейса пользователя для сохранения данных, SearchModels – для классов, в которых будут передаваться данные от интерфейса пользователя для поиска, и ViewModels – для классов, в которых будет передаваться информация пользователю. Папка с интерфейсами для описания работы с хранилищем данных (рисунок 1.3).

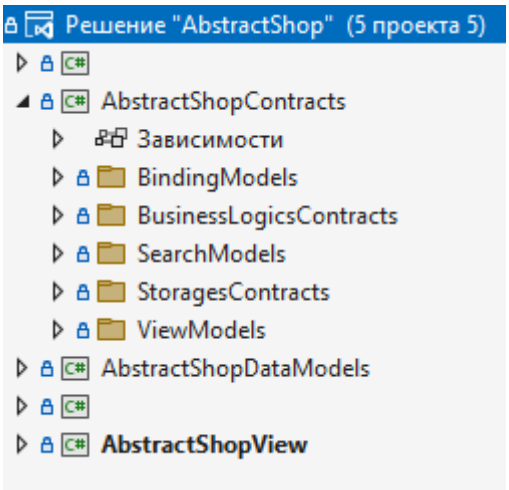

Рисунок 1.3 – Структура проекта AbstractShopContracts

Сделаем по сущностям классы с данными от пользователя (BindingModels). Классы будем называть по принципу «Имя сущности»BindingModel. В классах определим свойства, характерные для сущностей и объявленные в соответствующих интерфйесах-моделях (листинги 1.6-1.8).

```
using AbstractShopDataModels.Models;

namespace AbstractShopContracts.BindingModels
{
    public class ComponentBindingModel : IComponentModel
    {
        public int Id { get; set; }

        public string ComponentName { get; set; } = string.Empty;

        public double Cost { get; set; }
    }
}
```

Листинг 1.6 – Класс ComponentBindingModel

```
using AbstractShopDataModels.Models;

namespace AbstractShopContracts.BindingModels
{
    public class ProductBindingModel : IProductModel
    {
        public int Id { get; set; }

        public string ProductName { get; set; } = string.Empty;

        public double Price { get; set; }

        public Dictionary<int, (IComponentModel, int)> ProductComponents { get;
set; } = new();
    }
}
```

Листинг 1.7 – Класс ProductBindingModel

```
using AbstractShopDataModels.Enums;
using AbstractShopDataModels.Models;

namespace AbstractShopContracts.BindingModels
{
    public class OrderBindingModel : IOrderModel
    {
        public int Id { get; set; }

        public int ProductId { get; set; }

        public int Count { get; set; }

        public double Sum { get; set; }

        public OrderStatus Status { get; set; } = OrderStatus.Неизвестен;

        public DateTime DateCreate { get; set; } = DateTime.Now;
```

```
        public DateTime? DateImplement { get; set; }
    }
}
```

Листинг 1.8 – Класс OrderBindingModel

Далее модели для поиска (SearchModels). Называться они будут по аналогичному принципу: «Имя сущности»SearchModel. Но вот наполнение будет отличаться от Binding-моделей. Так как далеко не по каждому полю модели потребуется выполнять поиск. Например, для сущности «Компонент» нужен будет поиск по идентификатору и по имени, а по цене не требуется. Соответственно, в классе ComponentSearchModel будет только 2 поля (листинг 1.9). Для сущности «Изделия» поиск будет аналогичен по имени и по идентификатору (листинг 1.10). А для «Заказа» поиск будет пока только по идентификатору (листинг 1.11).

```
namespace AbstractShopContracts.SearchModels
{
    public class ComponentSearchModel
    {
        public int? Id { get; set; }

        public string? ComponentName { get; set; }
    }
}
```

Листинг 1.9 – Класс ComponentSearchModel

```
namespace AbstractShopContracts.SearchModels
{
    public class ProductSearchModel
    {
        public int? Id { get; set; }

        public string? ProductName { get; set; }
    }
}
```

Листинг 1.10 – Класс ProductSearchModel

```
namespace AbstractShopContracts.SearchModels
{
    public class OrderSearchModel
    {
        public int? Id { get; set; }
    }
                                        }
```

Листинг 1.11 – Класс OrderSearchModel

Следующий шаг – сделать классы с данными, передаваемые для отображения пользователю (ViewModels). Классы будем называть по

принципу «Имя сущности»ViewModel. Во многом они будут схожи с классами BindingModels (листинги 1.12-1.14).

```
using AbstractShopDataModels.Models;
using System.ComponentModel;

namespace AbstractShopContracts.ViewModels
{
    public class ComponentViewModel : IComponentModel
    {
        public int Id { get; set; }

        [DisplayName("Название компонента")]
        public string ComponentName { get; set; } = string.Empty;

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

Листинг 1.12 – Класс ComponentViewModel

```
using AbstractShopDataModels.Models;
using System.ComponentModel;

namespace AbstractShopContracts.ViewModels
{
    public class ProductViewModel : IProductModel
    {
        public int Id { get; set; }

        [DisplayName("Название изделия")]
        public string ProductName { get; set; } = string.Empty;

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

        public Dictionary<int, (IComponentModel, int)> ProductComponents { get;
set; } = new();
    }
}
```

Листинг 1.13 – Класс ProductViewModel

```
using AbstractShopDataModels.Enums;
using AbstractShopDataModels.Models;
using System.ComponentModel;

namespace AbstractShopContracts.ViewModels
{
    public class OrderViewModel : IOrderModel
    {
        [DisplayName("Номер")]
        public int Id { get; set; }

        public int ProductId { get; set; }

        [DisplayName("Изделие")]
        public string ProductName { get; set; } = string.Empty;

        [DisplayName("Количество")]
        public int Count { get; set; }
```

```
        [DisplayName("Сумма")]
        public double Sum { get; set; }

        [DisplayName("Статус")]
        public OrderStatus Status { get; set; } = OrderStatus.Неизвестен;

        [DisplayName("Дата создания")]
        public DateTime DateCreate { get; set; } = DateTime.Now;

        [DisplayName("Дата выполнения")]
        public DateTime? DateImplement { get; set; }
    }
}
```

Листинг 1.14 – Класс OrderViewModel

У некоторых свойств классов указан атрибут DisplayName. Он будет использоваться при выводе данных на форме в табличной форме.

Следующий шаг в проекте – создание интерфейсов. Какие методы должны быть объявлены в интерфейсе было описано выше, так что переходим сразу к созданию (листинги 1.15-1.17).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.ViewModels;

namespace AbstractShopContracts.BusinessLogicsContracts
{
    public interface IComponentLogic
    {
        List<ComponentViewModel>? ReadList(ComponentSearchModel? model);

        ComponentViewModel? ReadElement(ComponentSearchModel model);

        bool Create(ComponentBindingModel model);

        bool Update(ComponentBindingModel model);

        bool Delete(ComponentBindingModel model);
    }
}
```

Листинг 1.15 – Интерфейс IComponentLogic

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.ViewModels;

namespace AbstractShopContracts.BusinessLogicsContracts
{
    public interface IProductLogic
    {
        List<ProductViewModel>? ReadList(ProductSearchModel? model);

        ProductViewModel? ReadElement(ProductSearchModel model);

        bool Create(ProductBindingModel model);

        bool Update(ProductBindingModel model);
```

```
        bool Delete(ProductBindingModel model);
    }
                                        }
```

Листинг 1.16 – Интерфейс IProductLogic

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.ViewModels;

namespace AbstractShopContracts.BusinessLogicsContracts
{
    public interface IOrderLogic
    {
        List<OrderViewModel>? ReadList(OrderSearchModel? model);

        bool CreateOrder(OrderBindingModel model);

        bool TakeOrderInWork(OrderBindingModel model);

        bool FinishOrder(OrderBindingModel model);

        bool DeliveryOrder(OrderBindingModel model);
    }
}
```

Листинг 1.17 – Интерфейс IOrderLogic

Последнее – интерфейсы по хранилищу (листинги 1.18-1.20)

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.ViewModels;

namespace AbstractShopContracts.StoragesContracts
{
    public interface IComponentStorage
    {
        List<ComponentViewModel> GetFullList();

        List<ComponentViewModel> GetFilteredList(ComponentSearchModel model);

        ComponentViewModel? GetElement(ComponentSearchModel model);

        ComponentViewModel? Insert(ComponentBindingModel model);

        ComponentViewModel? Update(ComponentBindingModel model);

        ComponentViewModel? Delete(ComponentBindingModel model);
    }
}
```

Листинг 1.18 – Интерфейс IComponentStorage

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.ViewModels;

namespace AbstractShopContracts.StoragesContracts
{
    public interface IProductStorage
    {
        List<ProductViewModel> GetFullList();

        List<ProductViewModel> GetFilteredList(ProductSearchModel model);
```

```
        ProductViewModel? GetElement(ProductSearchModel model);

        ProductViewModel? Insert(ProductBindingModel model);

        ProductViewModel? Update(ProductBindingModel model);

        ProductViewModel? Delete(ProductBindingModel model);
    }
}
```

Листинг 1.19 – Интерфейс IProductStorage

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.ViewModels;

namespace AbstractShopContracts.StoragesContracts
{
    public interface IOrderStorage
    {
        List<OrderViewModel> GetFullList();

        List<OrderViewModel> GetFilteredList(OrderSearchModel model);

        OrderViewModel? GetElement(OrderSearchModel model);

        OrderViewModel? Insert(OrderBindingModel model);

        OrderViewModel? Update(OrderBindingModel model);

        OrderViewModel? Delete(OrderBindingModel model);
    }
}
```

Листинг 1.20 – Интерфейс IOrderStorage

Далее разработка может идти несколькими вариантами, за счет того, что были определены абстракции поведения. Можно сперва реализовать проект-представление, потом проект с бизнес-логикой, потом проект с хранилищем данных. Можно иначе, например, сперва сделать проект с бизнес-логикой, а потом остальные. Если разработка ведется командой, то можно вести разработку параллельно, кто-то занимается проектом с хранилищем, кто-то с представлением, а кто-то с бизнес-логикой и потом это все просто объединяется. Для примера, начнем с бизнес-логики.

Создадим проект для реализации бизнес-логики. Назовем проект «**AbstractShopBusinessLogic**». В ссылках у него добавим проект **AbstractShopContracts**. В проекте потребуется реализовать объявленные интерфейсы бизнес-логики. Классы с логикой для изделия и компонента будут простыми реализациями принципа CRUD (листинг 1.21).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.StoragesContracts;
using AbstractShopContracts.ViewModels;
using Microsoft.Extensions.Logging;

namespace AbstractShopBusinessLogic.BusinessLogics
{
    public class ComponentLogic : IComponentLogic
    {
        private readonly ILogger _logger;

        private readonly IComponentStorage _componentStorage;

        public ComponentLogic(ILogger<ComponentLogic> logger, IComponentStorage
componentStorage)
        {
            _logger = logger;
            _componentStorage = componentStorage;
        }

        public List<ComponentViewModel>? ReadList(ComponentSearchModel? model)
        {
            _logger.LogInformation("ReadList. ComponentName:{ComponentName}.
Id:{Id}", model?.ComponentName, model?.Id);
            var list = model == null ? _componentStorage.GetFullList() :
_componentStorage.GetFilteredList(model);
            if (list == null)
            {
                _logger.LogWarning("ReadList return null list");
                return null;
            }
            _logger.LogInformation("ReadList. Count:{Count}", list.Count);
            return list;
        }

        public ComponentViewModel? ReadElement(ComponentSearchModel model)
        {
            if (model == null)
            {
                throw new ArgumentNullException(nameof(model));
            }
            _logger.LogInformation("ReadElement. ComponentName:{ComponentName}.
Id:{Id}", model.ComponentName, model.Id);
            var element = _componentStorage.GetElement(model);
            if (element == null)
            {
                _logger.LogWarning("ReadElement element not found");
                return null;
            }
            _logger.LogInformation("ReadElement find. Id:{Id}", element.Id);
            return element;
        }

        public bool Create(ComponentBindingModel model)
        {
            CheckModel(model);
            if (_componentStorage.Insert(model) == null)
            {
                _logger.LogWarning("Insert operation failed");
                return false;
            }
            return true;
```

```
        }

        public bool Update(ComponentBindingModel model)
        {
            CheckModel(model);
            if (_componentStorage.Update(model) == null)
            {
                _logger.LogWarning("Update operation failed");
                return false;
            }
            return true;
        }

        public bool Delete(ComponentBindingModel model)
        {
            CheckModel(model, false);
            _logger.LogInformation("Delete. Id:{Id}", model.Id);
            if (_componentStorage.Delete(model) == null)
            {
                _logger.LogWarning("Delete operation failed");
                return false;
            }
            return true;
        }

        private void CheckModel(ComponentBindingModel model, bool withParams =
true)
        {
            if (model == null)
            {
                throw new ArgumentNullException(nameof(model));
            }
            if (!withParams)
            {
                return;
            }
            if (string.IsNullOrEmpty(model.ComponentName))
            {
                throw new ArgumentNullException("Нет названия компонента",
nameof(model.ComponentName));
            }
            if (model.Cost <= 0)
            {
                throw new ArgumentNullException("Цена компонента должна быть
больше 0", nameof(model.Cost));
            }
            _logger.LogInformation("Component. ComponentName:{ComponentName}.
Cost:{Cost}. Id:{Id}", model.ComponentName, model.Cost, model.Id);
            var element = _componentStorage.GetElement(new ComponentSearchModel
            {
                ComponentName = model.ComponentName
            });
            if (element != null && element.Id != model.Id)
            {
                throw new InvalidOperationException("Компонент с таким названием
уже есть");
            }
        }
    }
}
```

Листинг 1.21 – Класс ComponentLogic

Класс-реализация IProductLogic будет идентичным классу ComponentLogic, реализовать самостоятельно.

Класс с логикой для заказов будет отвечать за получение списка заказов, создания заказа и смены его статусов. Следует учитывать, что у заказа можно менять статус на новый, если его текущий статус предшествует новому (например, в статус «Готов» можно переводить, если заказ находится в статусе «Выполняется»). Класс реализовать самостоятельно.

Создадим проект «**AbstractShopListImplement**» для реализации интерфейсов хранения данных. В ссылках у него также добавим проекты **AbstractShopContracts** и «**AbstractShopDataModels**». В качестве хранилищ сущностей будут выступать списки (будем хранить только в операционной памяти, без сохранения на жестком диске). Для того, чтобы везде был один и тот же список сущностей сделаем класс-Singleton (паттерн Singleton). Нам потребуются классы-модели сущностей.

Класс-компонент будет задаваться на основе модели-интерфейса сущности «Компонент» (листинг 1.22). Чтобы защитить данные от неразрешенных манипуляций, метод set зададим приватным. В классе пропишем статический метод для создания объекта от класса-компонента на основе класса-bindingmodel, простой метод для изменения существующего объекта от класса-компонента на основе класса-bindingmodel и свойства с get-методом для создания объекта класса-viewmodel на основе данных объекта класса-компонента. Данный подход позволит в одном месте сосредоточить все манипуляции с полями сущности. Если в дальнейшем потребуется менять поля, то все изменения окажутся только в одном месте, что довольно удобно.

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.ViewModels;
using AbstractShopDataModels.Models;

namespace AbstractShopListImplement.Models
{
    public class Component : IComponentModel
    {
        public int Id { get; private set; }

        public string ComponentName { get; private set; } = string.Empty;
```

```
        public double Cost { get; set; }

        public static Component? Create(ComponentBindingModel? model)
        {
            if (model == null)
            {
                return null;
            }
            return new Component()
            {
                Id = model.Id,
                ComponentName = model.ComponentName,
                Cost = model.Cost
            };
        }

        public void Update(ComponentBindingModel? model)
        {
            if (model == null)
            {
                return;
            }
            ComponentName = model.ComponentName;
            Cost = model.Cost;
        }

        public ComponentViewModel GetViewModel => new()
        {
            Id = Id,
            ComponentName = ComponentName,
            Cost = Cost
        };
    }
}
```

Листинг 1.22 – Класс Component

Класс-изделие будет иметь схожую логику (листинг 1.23).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.ViewModels;
using AbstractShopDataModels.Models;

namespace AbstractShopListImplement.Models
{
    public class Product : IProductModel
    {
        public int Id { get; private set; }

        public string ProductName { get; private set; } = string.Empty;

        public double Price { get; private set; }

        public Dictionary<int, (IComponentModel, int)> ProductComponents { get;
private set; } = new Dictionary<int, (IComponentModel, int)>();

        public static Product? Create(ProductBindingModel? model)
        {
            if (model == null)
            {
                return null;
            }
            return new Product()
            {
```

```
                Id = model.Id,
                ProductName = model.ProductName,
                Price = model.Price,
                ProductComponents = model.ProductComponents
            };
        }

        public void Update(ProductBindingModel? model)
        {
            if (model == null)
            {
                return;
            }
            ProductName = model.ProductName;
            Price = model.Price;
            ProductComponents = model.ProductComponents;
        }

        public ProductViewModel GetViewModel => new()
        {
            Id = Id,
            ProductName = ProductName,
            Price = Price,
            ProductComponents = ProductComponents
        };
    }
}
```

Листинг 1.23 – Класс Product

Класс для модели сущности «Заказ» разработать самостоятельно. Теперь создадим класс для списков (листинг 1.24).

```
using AbstractShopListImplement.Models;

namespace AbstractShopListImplement
{
    public class DataListSingleton
    {
        private static DataListSingleton? _instance;

        public List<Component> Components { get; set; }

        public List<Order> Orders { get; set; }

        public List<Product> Products { get; set; }

        private DataListSingleton()
        {
            Components = new List<Component>();
            Orders = new List<Order>();
            Products = new List<Product>();
        }

        public static DataListSingleton GetInstance()
        {
            if (_instance == null)
            {
                _instance = new DataListSingleton();
            }

            return _instance;
        }
```

```
    }
}
```

Листинг 1.24 – Класс DataListSingleton

Как отмечалось ранее, класс будет реализовывать паттерн Singleton. И в нем будут храниться списки от всех сущностей.

Последний шаг – реализации интерфейсов. Начнем с реализации интерфейса IComponentStorage (листинг 1.25).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.StoragesContracts;
using AbstractShopContracts.ViewModels;
using AbstractShopListImplement.Models;

namespace AbstractShopListImplement.Implements
{
    public class ComponentStorage : IComponentStorage
    {
        private readonly DataListSingleton _source;

        public ComponentStorage()
        {
            _source = DataListSingleton.GetInstance();
        }

        public List<ComponentViewModel> GetFullList()
        {
            var result = new List<ComponentViewModel>();
            foreach (var component in _source.Components)
            {
                result.Add(component.GetViewModel);
            }
            return result;
        }

        public List<ComponentViewModel> GetFilteredList(ComponentSearchModel
model)
        {
            var result = new List<ComponentViewModel>();
            if (string.IsNullOrEmpty(model.ComponentName))
            {
                return result;
            }
            foreach (var component in _source.Components)
            {
                if (component.ComponentName.Contains(model.ComponentName))
                {
                    result.Add(component.GetViewModel);
                }
            }
            return result;
        }

        public ComponentViewModel? GetElement(ComponentSearchModel model)
        {
            if (string.IsNullOrEmpty(model.ComponentName) && !model.Id.HasValue)
            {
                return null;
            }
```

```
            foreach (var component in _source.Components)
            {
                if ((!string.IsNullOrEmpty(model.ComponentName) &&
component.ComponentName == model.ComponentName) ||
                    (model.Id.HasValue && component.Id == model.Id))
                {
                    return component.GetViewModel;
                }
            }
            return null;
        }

        public ComponentViewModel? Insert(ComponentBindingModel model)
        {
            model.Id = 1;
            foreach (var component in _source.Components)
            {
                if (model.Id <= component.Id)
                {
                    model.Id = component.Id + 1;
                }
            }
            var newComponent = Component.Create(model);
            if (newComponent == null)
            {
                return null;
            }
            _source.Components.Add(newComponent);
            return newComponent.GetViewModel;
        }

        public ComponentViewModel? Update(ComponentBindingModel model)
        {
            foreach (var component in _source.Components)
            {
                if (component.Id == model.Id)
                {
                    component.Update(model);
                    return component.GetViewModel;
                }
            }
            return null;
        }

        public ComponentViewModel? Delete(ComponentBindingModel model)
        {
            for (int i = 0; i < _source.Components.Count; ++i)
            {
                if (_source.Components[i].Id == model.Id)
                {
                    var element = _source.Components[i];
                    _source.Components.RemoveAt(i);
                    return element.GetViewModel;
                }
            }
            return null;
        }
    }
}
```

Листинг 1.25 – Класс ComponentStorage

Для работы со списком будет поле source от класса DataListSingleton (в конструкторе получаем сам объект). При создании также требуется определить новый Id для компонента (ищем максимальный Id и прибавляем 1).

Реализация интерфейсов IProductStorage и IOrderStorage будет аналогичной. Классы реализовать самостоятельно.

Переходим в проект «**AbstractShopView**». Первым делом введем IoC-контейнер для установления зависимостей между интерфейсами их реализациями и добавим ссылку на созданный проекты с бизнес-логикой и хранилищем. Будем использовать тот же подход, что и в прошлом семестре через ServiceCollection и ServiceProvider.

Далее создадим формы. Начнем с компонента. Сделаем форму для создания/изменения компонента. Там будет все просто, два поля для ввода названия компонента и его цены (рисунок 1.4).

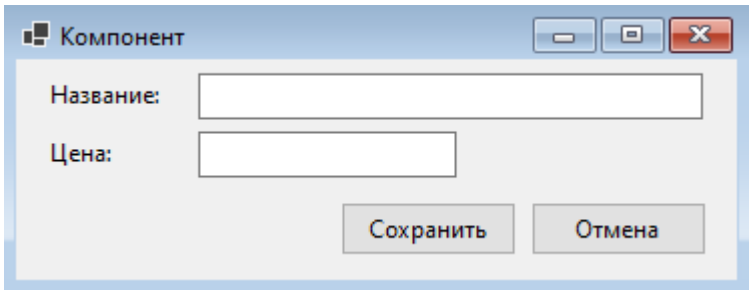

Рисунок 1.4 – Форма работы с компонентом

Логика формы будет следующей (листинг 1.26).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.SearchModels;
using Microsoft.Extensions.Logging;

namespace AbstractShopView
{
    public partial class FormComponent : Form
    {
        private readonly ILogger _logger;

        private readonly IComponentLogic _logic;

        private int? _id;

        public int Id { set { _id = value; } }

        public FormComponent(ILogger<FormComponent> logger, IComponentLogic
logic)
        {
            InitializeComponent();
```

```
            _logger = logger;
            _logic = logic;
        }

        private void FormComponent_Load(object sender, EventArgs e)
        {
            if (_id.HasValue)
            {
                try
                {
                    _logger.LogInformation("Получение компонента");
                    var view = _logic.ReadElement(new ComponentSearchModel { Id =
_id.Value });
                    if (view != null)
                    {
                        textBoxName.Text = view.ComponentName;
                        textBoxCost.Text = view.Cost.ToString();
                    }
                }
                catch (Exception ex)
                {
                    _logger.LogError(ex, "Ошибка получения компонента");
                    MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                }
            }
        }

        private void ButtonSave_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(textBoxName.Text))
            {
                MessageBox.Show("Заполните название", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            _logger.LogInformation("Сохранение компонента");
            try
            {
                var model = new ComponentBindingModel
                {
                    Id = _id ?? 0,
                    ComponentName = textBoxName.Text,
                    Cost = Convert.ToDouble(textBoxCost.Text)
                };
                var operationResult = _id.HasValue ? _logic.Update(model) :
_logic.Create(model);
                if (!operationResult)
                {
                    throw new Exception("Ошибка при сохранении. Дополнительная
информация в логах.");
                }
                MessageBox.Show("Сохранение прошло успешно", "Сообщение",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                DialogResult = DialogResult.OK;
                Close();
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка сохранения компонента");
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }
```

```
        private void ButtonCancel_Click(object sender, EventArgs e)
        {
            DialogResult = DialogResult.Cancel;
            Close();
        }
    }
}
```

Листинг 1.26 – Логика формы FormComponent

В конструкторе будем передавать объект класса ComponentLogic (в его конструкторе ServiceProvider будет подставлять реализацию интерфейса IComponentStorage). В методе загрузки будем проверять, если поле id заполнено, то пытаться получить запись и вывести ее на экран. При сохранении, передаем данные для добавления или редактирования записи. При отмене просто закрываем форму.

Далее сделаем форму для вывода всех имеющихся компонент (рисунок 1.5).

Логика формы будет следующей (листинг 1.27).

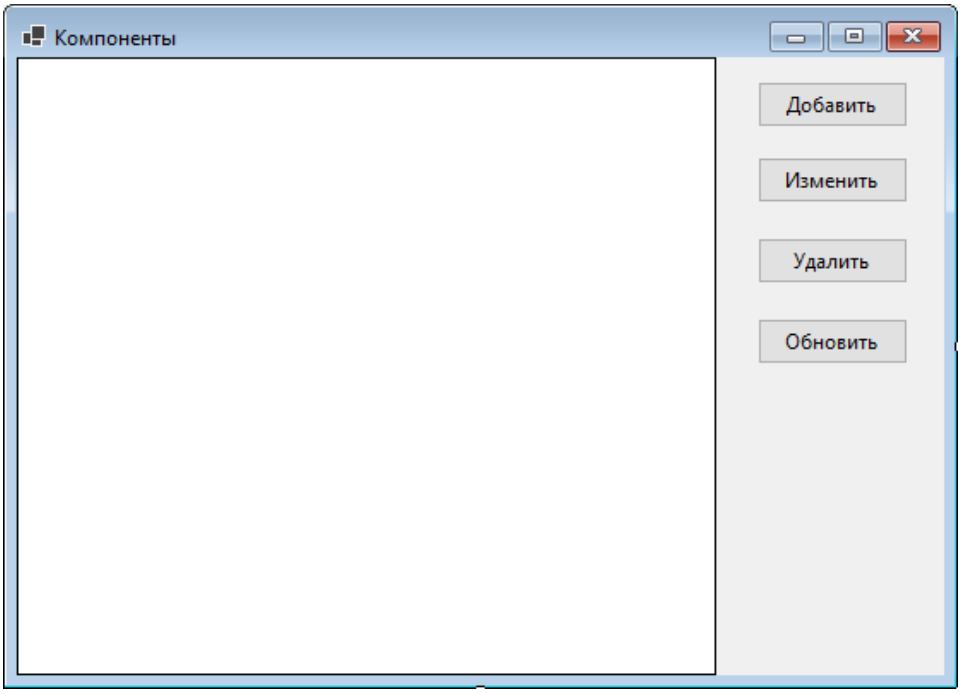

Рисунок 1.5 – Форма работы с набором компонент

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using Microsoft.Extensions.Logging;

namespace AbstractShopView
{
    public partial class FormComponents : Form
    {
```

```
        private readonly ILogger _logger;

        private readonly IComponentLogic _logic;

        public FormComponents(ILogger<FormComponents> logger, IComponentLogic
logic)
        {
            InitializeComponent();
            _logger = logger;
            _logic = logic;
        }

        private void FormComponents_Load(object sender, EventArgs e)
        {
            LoadData();
        }

        private void LoadData()
        {
            try
            {
                var list = _logic.ReadList(null);
                if (list != null)
                {
                    dataGridView.DataSource = list;
                    dataGridView.Columns["Id"].Visible = false;
                    dataGridView.Columns["ComponentName"].AutoSizeMode =
DataGridViewAutoSizeColumnMode.Fill;
                }
                _logger.LogInformation("Загрузка компонентов");
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка загрузки компонентов");
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }

        private void ButtonAdd_Click(object sender, EventArgs e)
        {
            var service =
Program.ServiceProvider?.GetService(typeof(FormComponent));
            if (service is FormComponent form)
            {
                if (form.ShowDialog() == DialogResult.OK)
                {
                    LoadData();
                }
            }
        }

        private void ButtonUpd_Click(object sender, EventArgs e)
        {
            if (dataGridView.SelectedRows.Count == 1)
            {
                var service =
Program.ServiceProvider?.GetService(typeof(FormComponent));
                if (service is FormComponent form)
                {
                    form.Id =
Convert.ToInt32(dataGridView.SelectedRows[0].Cells["Id"].Value);
                    if (form.ShowDialog() == DialogResult.OK)
                    {
```

```
                        LoadData();
                    }
                }
            }
        }

        private void ButtonDel_Click(object sender, EventArgs e)
        {
            if (dataGridView.SelectedRows.Count == 1)
            {
                if (MessageBox.Show("Удалить запись?", "Вопрос",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
                {
                    int id =
Convert.ToInt32(dataGridView.SelectedRows[0].Cells["Id"].Value);
                    _logger.LogInformation("Удаление компонента");
                    try
                    {
                        if (!_logic.Delete(new ComponentBindingModel { Id = id
}))
                        {
                            throw new Exception("Ошибка при удалении.
Дополнительная информация в логах.");
                        }
                        LoadData();
                    }
                    catch (Exception ex)
                    {
                        _logger.LogError(ex, "Ошибка удаления компонента");
                        MessageBox.Show(ex.Message, "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    }
                }
            }
        }

        private void ButtonRef_Click(object sender, EventArgs e)
        {
            LoadData();
        }
    }
}
```

Листинг 1.27 – Логика формы FormComponents

При получении списка будет приходить список объектов ComponentViewModel. Список будем передавать в элемент DataGridView. Так как, каждый объект содержит 3 поля: id, название компонента и цена, то в DataGridView создается 3 колонки под эти поля. Id нужен для внутренних нужд и его не требуется выводить пользователю. Скроем первую колонку в DataGridView. А для второй колонки настроем отображение на всю доступную ширину элемента DataGridView. В логике добавления и изменения будем вызывать форму для работы с компонентом, используя ServiceProvider. В логике изменения и удаления в первую очередь проверяем, что есть выбранная

строка и вытаскиваем значение из первой ячейки выбранной строки. В конце логики для всех кнопок вызывается метод обновления списка.

Для изделий потребуется три формы: для списка, для отдельного изделия (создание и редактирование) и для добавления компонент в изделие. Начнем с последней формы.

Нам потребуется выбирать компонент и указывать в каком количестве он будет использоваться в изделии. Для выбора компонента на форму поместим элемент comboBox, для указания количества – textBox (рисунок 1.6).

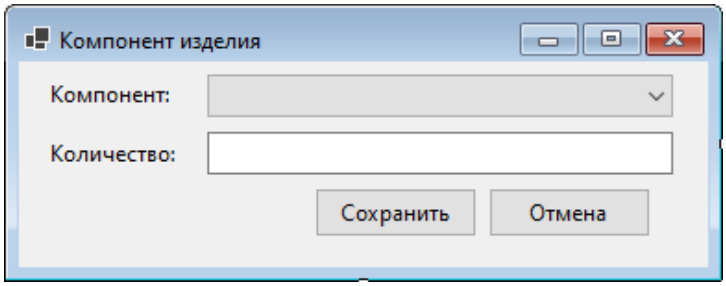

Рисунок 1.6 – Форма выбора компонента в изделие

Логика формы будет следующей (листинг 1.28).

```
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.ViewModels;
using AbstractShopDataModels.Models;

namespace AbstractShopView
{
    public partial class FormProductComponent : Form
    {
        private readonly List<ComponentViewModel>? _list;

        public int Id { get { return
Convert.ToInt32(comboBoxComponent.SelectedValue); } set {
comboBoxComponent.SelectedValue = value; } }

        public IComponentModel? ComponentModel
        {
            get
            {
                if (_list == null)
                {
                    return null;
                }
                foreach(var elem in _list)
                {
                    if (elem.Id == Id)
                    {
                        return elem;
                    }
                }
                return null;
            }
        }
```

```
        public int Count { get { return Convert.ToInt32(textBoxCount.Text); } set
{ textBoxCount.Text = value.ToString(); } }

        public FormProductComponent(IComponentLogic logic)
        {
            InitializeComponent();

            _list = logic.ReadList(null);
            if (_list != null)
            {
                comboBoxComponent.DisplayMember = "ComponentName";
                comboBoxComponent.ValueMember = "Id";
                comboBoxComponent.DataSource = _list;
                comboBoxComponent.SelectedItem = null;
            }
        }

        private void ButtonSave_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(textBoxCount.Text))
            {
                MessageBox.Show("Заполните поле Количество", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            if (comboBoxComponent.SelectedValue == null)
            {
                MessageBox.Show("Выберите компонент", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            DialogResult = DialogResult.OK;
            Close();
        }

        private void ButtonCancel_Click(object sender, EventArgs e)
        {
            DialogResult = DialogResult.Cancel;
            Close();
        }
    }
}
```

Листинг 1.28 – Логика формы FormProductComponent

В логике будут свойства для получения/возврата информации о компоненте и количестве. В конструкторе формы получаем список компонент и заносим его в выпадающий список (так как в свойстве Id прописали работу сразу с comboBoxComponent, то он должен быть заполнен до обращения к этому свойству, а не в момент загрузки формы). При сохранении проверяем, что поля заполнены и закрываем форму.

Перейдем к форме Изделия. У изделия требуется вводить название, стоимость, а также список компонентов, которые в нем используются. Потому

форма будет чуть сложнее, чем у компонента (рисунок 1.7). Для вывода компонент будет использовать DataGridView, у которого ручками добавим колонки (3 колонки для id компонента, которая будет скрытая, названия и количества).

Логика формы будет следующей (листинг 1.29).

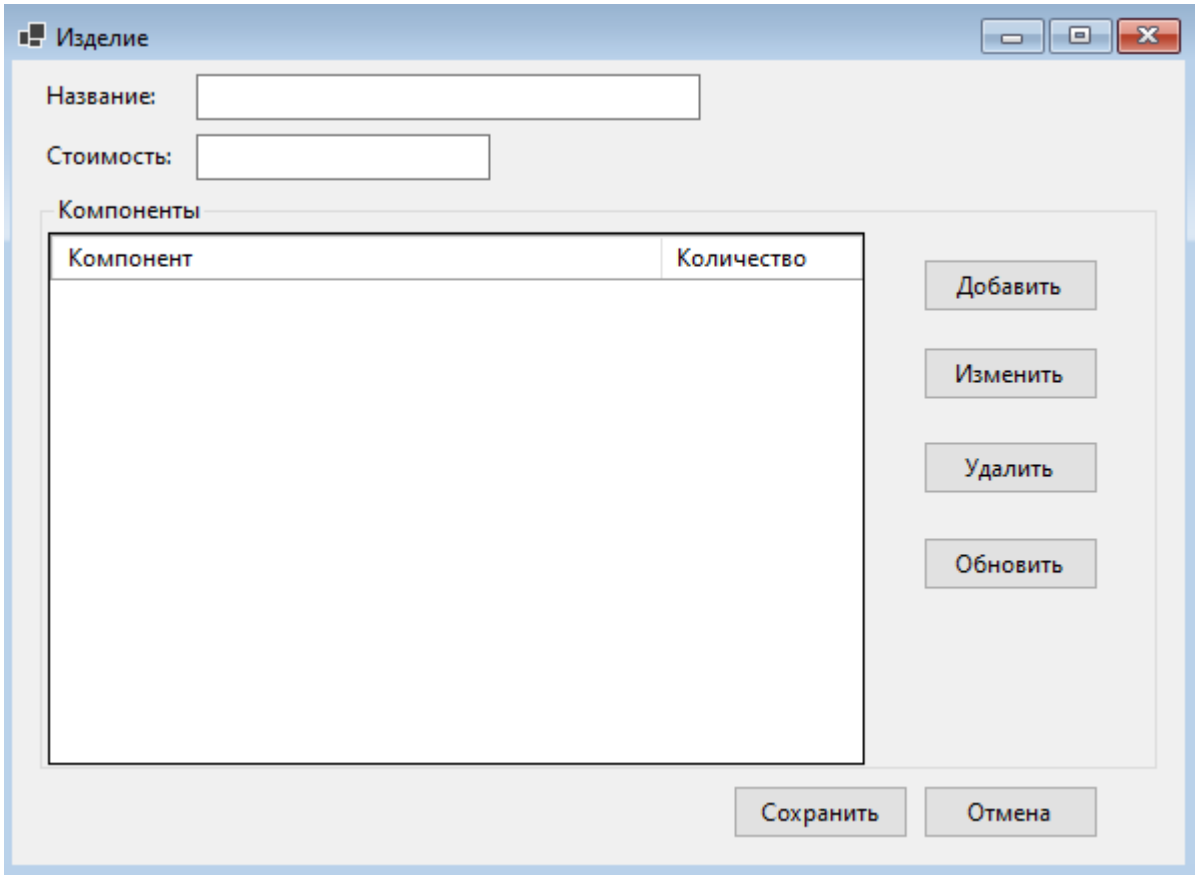

Рисунок 1.7 – Форма работы с изделием

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.SearchModels;
using AbstractShopDataModels.Models;
using Microsoft.Extensions.Logging;

namespace AbstractShopView
{
    public partial class FormProduct : Form
    {
        private readonly ILogger _logger;

        private readonly IProductLogic _logic;

        private int? _id;

        private Dictionary<int, (IComponentModel, int)> _productComponents;

        public int Id { set { _id = value; } }

        public FormProduct(ILogger<FormProduct> logger, IProductLogic logic)
```

```
        {
            InitializeComponent();
            _logger = logger;
            _logic = logic;
            _productComponents = new Dictionary<int, (IComponentModel, int)>();
        }

        private void FormProduct_Load(object sender, EventArgs e)
        {
            if (_id.HasValue)
            {
                _logger.LogInformation("Загрузка изделия");
                try
                {
                    var view = _logic.ReadElement(new ProductSearchModel { Id =
_id.Value });
                    if (view != null)
                    {
                        textBoxName.Text = view.ProductName;
                        textBoxPrice.Text = view.Price.ToString();
                        _productComponents = view.ProductComponents ?? new
Dictionary<int, (IComponentModel, int)>();
                        LoadData();
                    }
                }
                catch (Exception ex)
                {
                    _logger.LogError(ex, "Ошибка загрузки изделия");
                    MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                }
            }
        }

        private void LoadData()
        {
            _logger.LogInformation("Загрузка компонент изделия");
            try
            {
                if (_productComponents != null)
                {
                    dataGridView.Rows.Clear();
                    foreach (var pc in _productComponents)
                    {
                        dataGridView.Rows.Add(new object[] { pc.Key,
pc.Value.Item1.ComponentName, pc.Value.Item2 });
                    }
                    textBoxPrice.Text = CalcPrice().ToString();
                }
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка загрузки компонент изделия");
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }

        private void ButtonAdd_Click(object sender, EventArgs e)
        {
            var service =
Program.ServiceProvider?.GetService(typeof(FormProductComponent));
            if (service is FormProductComponent form)
            {
```

```
                if (form.ShowDialog() == DialogResult.OK)
                {
                    if (form.ComponentModel == null)
                    {
                        return;
                    }
                    _logger.LogInformation("Добавление нового компонента:
{ComponentName} - {Count}", form.ComponentModel.ComponentName, form.Count);
                    if (_productComponents.ContainsKey(form.Id))
                    {
                        _productComponents[form.Id] = (form.ComponentModel,
form.Count);
                    }
                    else
                    {
                        _productComponents.Add(form.Id, (form.ComponentModel,
form.Count));
                    }
                    LoadData();
                }
            }
        }

        private void ButtonUpd_Click(object sender, EventArgs e)
        {
            if (dataGridView.SelectedRows.Count == 1)
            {
                var service =
Program.ServiceProvider?.GetService(typeof(FormProductComponent));
                if (service is FormProductComponent form)
                {
                    int id =
Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);
                    form.Id = id;
                    form.Count = _productComponents[id].Item2;
                    if (form.ShowDialog() == DialogResult.OK)
                    {
                        if (form.ComponentModel == null)
                        {
                            return;
                        }
                        _logger.LogInformation("Изменение компонента:
{ComponentName} - {Count}", form.ComponentModel.ComponentName, form.Count);
                        _productComponents[form.Id] = (form.ComponentModel,
form.Count);
                        LoadData();
                    }
                }
            }
        }

        private void ButtonDel_Click(object sender, EventArgs e)
        {
            if (dataGridView.SelectedRows.Count == 1)
            {
                if (MessageBox.Show("Удалить запись?", "Вопрос",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
                {
                    try
                    {
                        _logger.LogInformation("Удаление компонента:
{ComponentName} - {Count}", dataGridView.SelectedRows[0].Cells[1].Value);
```

```
_productComponents?.Remove(Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].
Value));
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show(ex.Message, "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    }
                    LoadData();
                }
            }
        }

        private void ButtonRef_Click(object sender, EventArgs e)
        {
            LoadData();
        }

        private void ButtonSave_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(textBoxName.Text))
            {
                MessageBox.Show("Заполните название", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            if (string.IsNullOrEmpty(textBoxPrice.Text))
            {
                MessageBox.Show("Заполните цену", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                return;
            }
            if (_productComponents == null || _productComponents.Count == 0)
            {
                MessageBox.Show("Заполните компоненты", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            _logger.LogInformation("Сохранение изделия");
            try
            {
                var model = new ProductBindingModel
                {
                    Id = _id ?? 0,
                    ProductName = textBoxName.Text,
                    Price = Convert.ToDouble(textBoxPrice.Text),
                    ProductComponents = _productComponents
                };
                var operationResult = _id.HasValue ? _logic.Update(model) :
_logic.Create(model);
                if (!operationResult)
                {
                    throw new Exception("Ошибка при сохранении. Дополнительная
информация в логах.");
                }
                MessageBox.Show("Сохранение прошло успешно", "Сообщение",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                DialogResult = DialogResult.OK;
                Close();
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка сохранения изделия");
```

```
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }

        private void ButtonCancel_Click(object sender, EventArgs e)
        {
            DialogResult = DialogResult.Cancel;
            Close();
        }

        private double CalcPrice()
        {
            double price = 0;
            foreach (var elem in _productComponents)
            {
                price += ((elem.Value.Item1?.Cost ?? 0) * elem.Value.Item2);
            }
            return Math.Round(price * 1.1, 2);
        }
    }
}
```

Листинг 1.29 – Логика формы FormProduct

Здесь будет комбинация из логики работы с элементом (изделие) и списком (компоненты изделия). В отдельном поле будем хранить список компонент изделия и выводить его в DataGridView.

Форма и логика для списка изделий будет идентична форме списка компонент.

Остается форма создания заказа и главная форма. Начнем с формы создания заказа. При создании заказа потребуется указать изделие (выпадающий список), количество и сумму (должна высчитываться из цены изделия и количества) (рисунок 1.8).

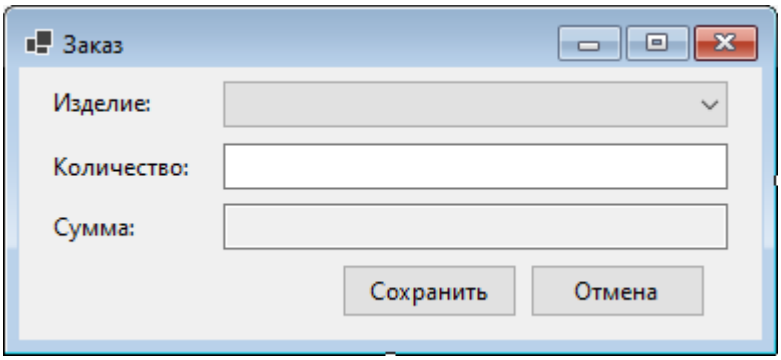

Рисунок 1.8 – Форма создания заказа

Логика формы будет следующей (листинг 1.30).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.SearchModels;
```

```
using Microsoft.Extensions.Logging;

namespace AbstractShopView
{
    public partial class FormCreateOrder : Form
    {
        private readonly ILogger _logger;

        private readonly IProductLogic _logicP;

        private readonly IOrderLogic _logicO;

        public FormCreateOrder(ILogger<FormCreateOrder> logger, IProductLogic
logicP, IOrderLogic logicO)
        {
            InitializeComponent();
            _logger = logger;
            _logicP = logicP;
            _logicO = logicO;
        }

        private void FormCreateOrder_Load(object sender, EventArgs e)
        {
            _logger.LogInformation("Загрузка изделий для заказа");
            // прописать логику
        }

        private void CalcSum()
        {
            if (comboBoxProduct.SelectedValue != null &&
!string.IsNullOrEmpty(textBoxCount.Text))
            {
                try
                {
                    int id = Convert.ToInt32(comboBoxProduct.SelectedValue);
                    var product = _logicP.ReadElement(new ProductSearchModel { Id
= id });
                    int count = Convert.ToInt32(textBoxCount.Text);
                    textBoxSum.Text = Math.Round(count * (product?.Price ?? 0),
2).ToString();
                    _logger.LogInformation("Расчет суммы заказа");
                }
                catch (Exception ex)
                {
                    _logger.LogError(ex, "Ошибка расчета суммы заказа");
                    MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                }
            }
        }

        private void TextBoxCount_TextChanged(object sender, EventArgs e)
        {
            CalcSum();
        }

        private void ComboBoxProduct_SelectedIndexChanged(object sender,
EventArgs e)
        {
            CalcSum();
        }

        private void ButtonSave_Click(object sender, EventArgs e)
        {
```

```
            if (string.IsNullOrEmpty(textBoxCount.Text))
            {
                MessageBox.Show("Заполните поле Количество", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            if (comboBoxProduct.SelectedValue == null)
            {
                MessageBox.Show("Выберите изделие", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            _logger.LogInformation("Создание заказа");
            try
            {
                var operationResult = _logicO.CreateOrder(new OrderBindingModel
                {
                    ProductId = Convert.ToInt32(comboBoxProduct.SelectedValue),
                    Count = Convert.ToInt32(textBoxCount.Text),
                    Sum = Convert.ToDouble(textBoxSum.Text)
                });
                if (!operationResult)
                {
                    throw new Exception("Ошибка при создании заказа.
Дополнительная информация в логах.");
                }
                MessageBox.Show("Сохранение прошло успешно", "Сообщение",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                DialogResult = DialogResult.OK;
                Close();
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка создания заказа");
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }

        private void ButtonCancel_Click(object sender, EventArgs e)
        {
            DialogResult = DialogResult.Cancel;
            Close();
        }
    }
}
```

Листинг 1.30 – Логика формы FormCreateOrder

При загрузке формы подгружаем список изделий. Делаем отдельный метод расчета суммы и вызываем его при изменении в поле «Количество» или при выборе элемента из выпадающего списка.

И последняя форма – главная форма приложения. В ней будет отображаться список всех заказов, методы для создания заказов, смены статусов, а также для доступа к спискам компонент и изделий (сделаем через пункты меню) (рисунок 1.9).

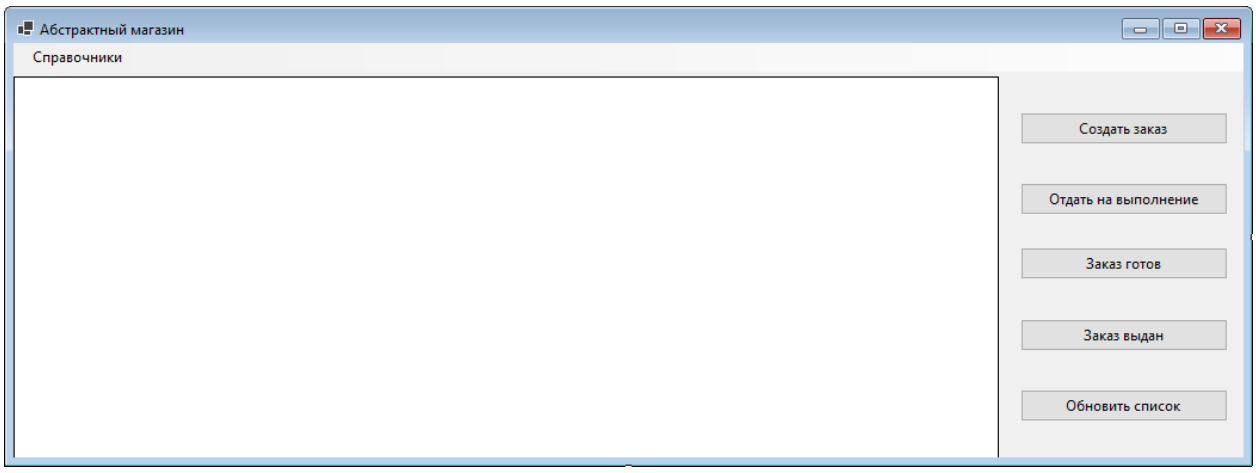

Рисунок 1.9 – Главная форма

Логика формы будет следующей (листинг 1.31).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using Microsoft.Extensions.Logging;

namespace AbstractShopView
{
    public partial class FormMain : Form
    {
        private readonly ILogger _logger;

        private readonly IOrderLogic _orderLogic;

        public FormMain(ILogger<FormMain> logger, IOrderLogic orderLogic)
        {
            InitializeComponent();
            _logger = logger;
            _orderLogic = orderLogic;
        }

        private void FormMain_Load(object sender, EventArgs e)
        {
            LoadData();
        }

        private void LoadData()
        {
            _logger.LogInformation("Загрузка заказов");
            // прописать логику
        }

        private void КомпонентыToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            var service =
Program.ServiceProvider?.GetService(typeof(FormComponents));
            if (service is FormComponents form)
            {
                form.ShowDialog();
            }
        }

        private void ИзделияToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // прописать логику
```

```
        }

        private void ButtonCreateOrder_Click(object sender, EventArgs e)
        {
            var service =
Program.ServiceProvider?.GetService(typeof(FormCreateOrder));
            if (service is FormCreateOrder form)
            {
                form.ShowDialog();
                LoadData();
            }
        }

        private void ButtonTakeOrderInWork_Click(object sender, EventArgs e)
        {
            if (dataGridView.SelectedRows.Count == 1)
            {
                int id =
Convert.ToInt32(dataGridView.SelectedRows[0].Cells["Id"].Value);
                _logger.LogInformation("Заказ №{id}. Меняется статус на 'В
работе'", id);
                try
                {
                    var operationResult = _orderLogic.TakeOrderInWork(new
OrderBindingModel { Id = id });
                    if (!operationResult)
                    {
                        throw new Exception("Ошибка при сохранении.
Дополнительная информация в логах.");
                    }
                    LoadData();
                }
                catch (Exception ex)
                {
                    _logger.LogError(ex, "Ошибка передачи заказа в работу");
                    MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                }
            }
        }

        private void ButtonOrderReady_Click(object sender, EventArgs e)
        {
            if (dataGridView.SelectedRows.Count == 1)
            {
                int id =
Convert.ToInt32(dataGridView.SelectedRows[0].Cells["Id"].Value);
                _logger.LogInformation("Заказ №{id}. Меняется статус на 'Готов'",
id);
                try
                {
                    var operationResult = _orderLogic.FinishOrder(new
OrderBindingModel { Id = id });
                    if (!operationResult)
                    {
                        throw new Exception("Ошибка при сохранении.
Дополнительная информация в логах.");
                    }
                    LoadData();
                }
                catch (Exception ex)
                {
                    _logger.LogError(ex, "Ошибка отметки о готовности заказа");
```

```
                    MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                }
            }
        }

        private void ButtonIssuedOrder_Click(object sender, EventArgs e)
        {
            if (dataGridView.SelectedRows.Count == 1)
            {
                int id =
Convert.ToInt32(dataGridView.SelectedRows[0].Cells["Id"].Value);
                _logger.LogInformation("Заказ №{id}. Меняется статус на 'Выдан'",
id);
                try
                {
                    var operationResult = _orderLogic.DeliveryOrder(new
OrderBindingModel { Id = id });
                    if (!operationResult)
                    {
                        throw new Exception("Ошибка при сохранении.
Дополнительная информация в логах.");
                    }
                    _logger.LogInformation("Заказ №{id} выдан", id);
                    LoadData();
                }
                catch (Exception ex)
                {
                    _logger.LogError(ex, "Ошибка отметки о выдачи заказа");
                    MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                }
            }
        }

        private void ButtonRef_Click(object sender, EventArgs e)
        {
            LoadData();
        }
    }
}
```

Листинг 1.31 – Логика формы FormMain

Для вызова форм списков компонент и изделий, а также формы создания заказа используется ServiceProvider. Для смены статусов логика простая, проверяем, что есть выбранная строка, извлекаем идентификатор заказа и вызываем нужный метод.

Останется только изменить вызов главной формы в классе Program. Однако, для этого требуется сперва создать реализации для интерфейсов бизнес-логики и настроить ServiceProvider. Потому перейдем к реализации.

В классе Program настраиваем ServiceProvider и вызываем главную форму при запуске программы (листинг 1.32).

```
using AbstractShopBusinessLogic.BusinessLogics;
```

```
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.StoragesContracts;
using AbstractShopListImplement.Implements;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using NLog.Extensions.Logging;

namespace AbstractShopView
{
    internal static class Program
    {
            private static ServiceProvider? _serviceProvider;
            public static ServiceProvider? ServiceProvider => _serviceProvider;
            /// <summary>
            ///  The main entry point for the application.
            /// </summary>
            [STAThread]
        static void Main()
        {
            // To customize application configuration such as set high DPI
settings or default font,
            // see https://aka.ms/applicationconfiguration.
            ApplicationConfiguration.Initialize();
            var services = new ServiceCollection();
            ConfigureServices(services);
                   _serviceProvider = services.BuildServiceProvider();

      Application.Run(_serviceProvider.GetRequiredService<FormMain>());
            }

            private static void ConfigureServices(ServiceCollection services)
            {
                   services.AddLogging(option =>
                                {

      option.SetMinimumLevel(LogLevel.Information);
                                       option.AddNLog("nlog.config");
                                });
                   services.AddTransient<IComponentStorage, ComponentStorage>();
                   services.AddTransient<IOrderStorage, OrderStorage>();
                   services.AddTransient<IProductStorage, ProductStorage>();

                   services.AddTransient<IComponentLogic, ComponentLogic>();
                   services.AddTransient<IOrderLogic, OrderLogic>();
                   services.AddTransient<IProductLogic, ProductLogic>();

                   services.AddTransient<FormMain>();
                   services.AddTransient<FormComponent>();
                   services.AddTransient<FormComponents>();
                   services.AddTransient<FormCreateOrder>();
                   services.AddTransient<FormProduct>();
                   services.AddTransient<FormProductComponent>();
                   services.AddTransient<FormProducts>();
            }
      }
}
```

Листинг 1.32 – Класс Program

Контрольный пример

Главная форма представлена на рисунке 1.10.

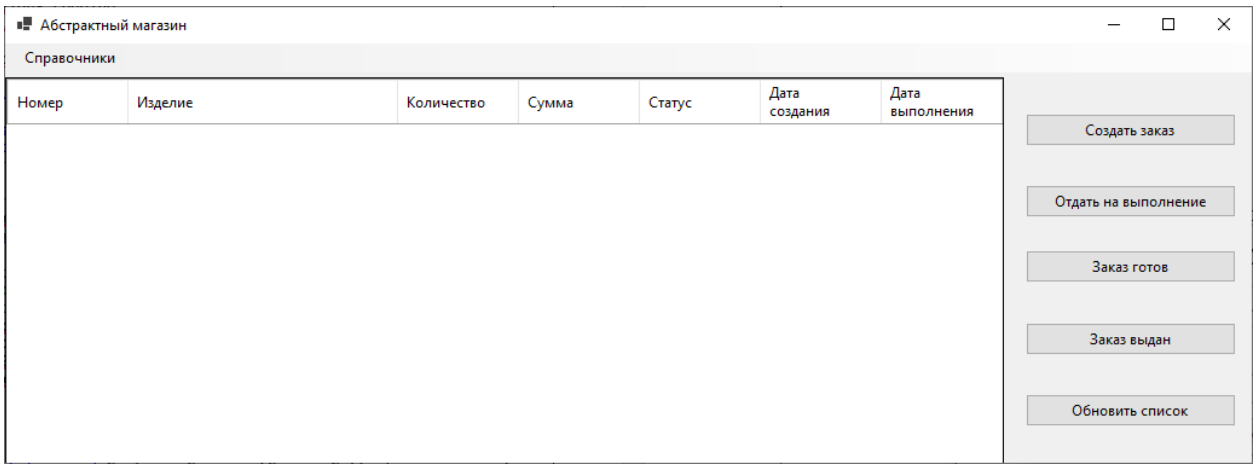

Рисунок 1.10 – Главная форма

Создание компонента представлено на рисунке 1.11.

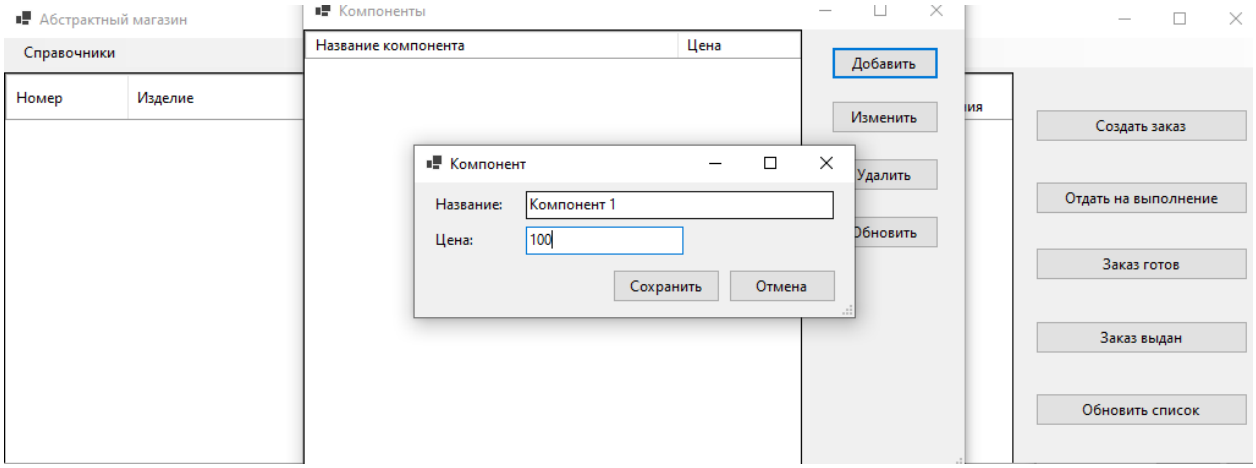

Рисунок 1.11 – Создание компонента

Создание изделия и добавление компонента к изделию представлено на рисунке 1.12.

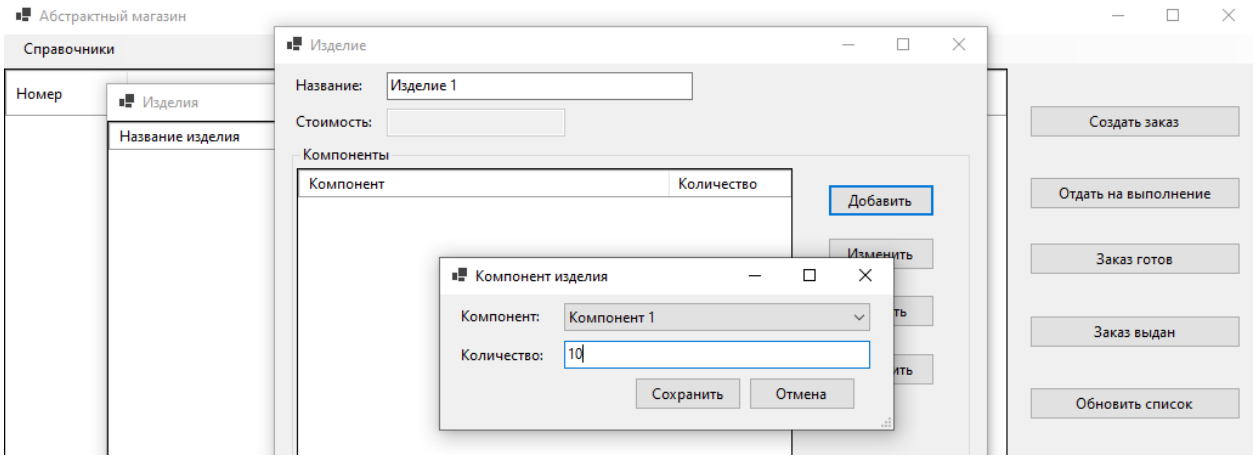

Рисунок 1.12 – Добавление компонента к изделию

Создание заказа представлено на рисунке 1.13.

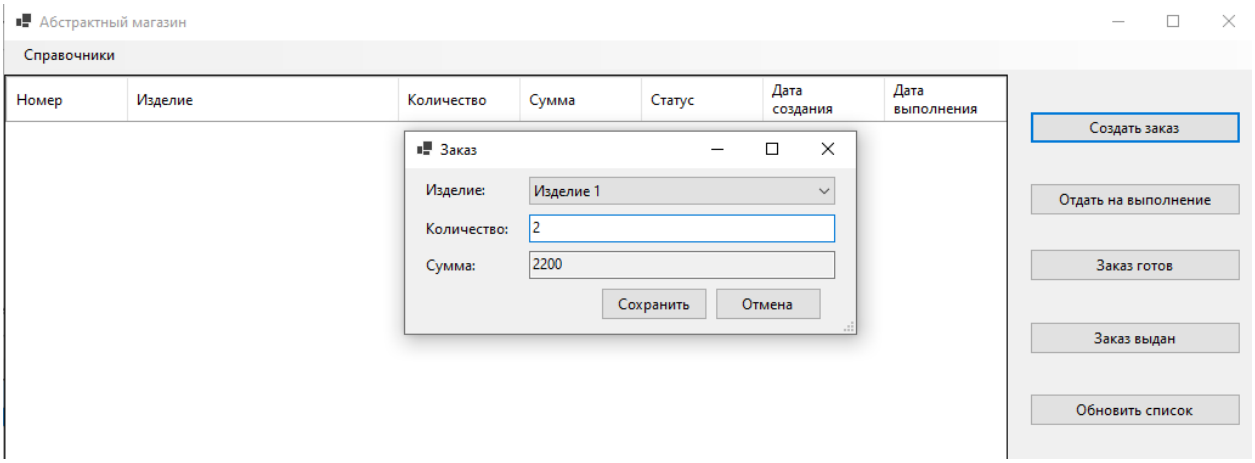

Рисунок 1.13 – Создание заказа

Проверка на неверный статус представлено на рисунке 1.14.

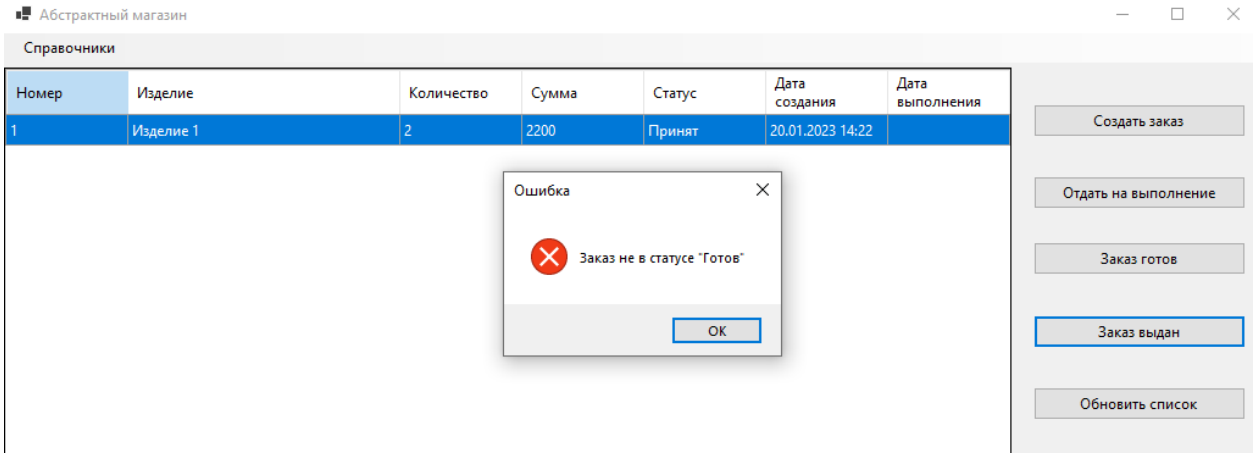

Рисунок 1.14 – Проверка на неверный статус

Правильный результат смены статуса представлен на рисунке 1.15.

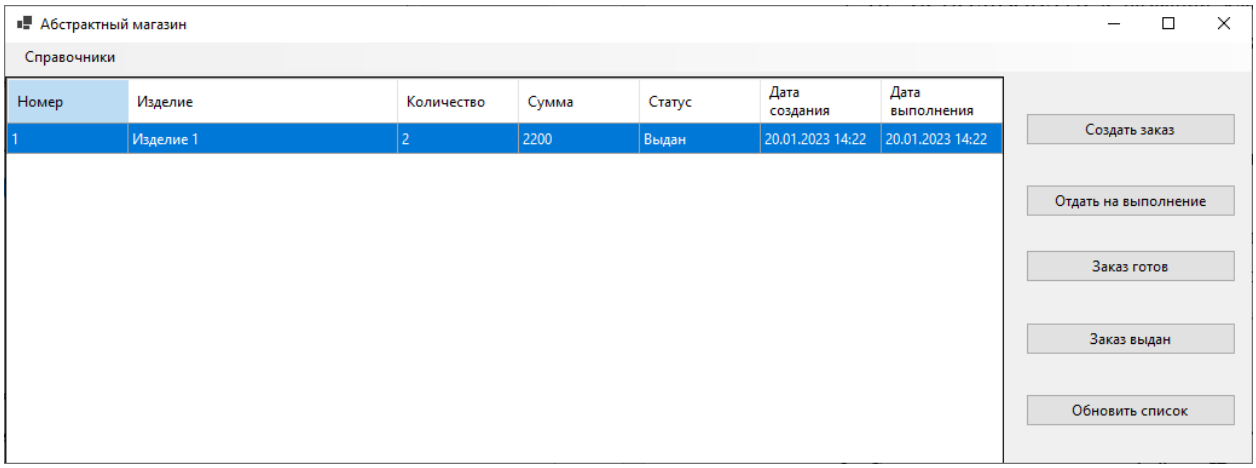

Рисунок 1.15 – Заказ со статусом «Выдан»

Требования

1. Название проектов должны ОТЛИЧАТЬСЯ от названия проектов, приведенных в примере и должны соответствовать логике вашего задания по варианту.
2. Название форм, классов, свойств классов должно соответствовать логике вашего задания по варианту.
3. НЕ ИСПОЛЬЗОВАТЬ в названии класса, связанного с изделием слово «Product» (во вариантах в скобках указано название класса для изделия)!!!
4. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).
5. Создать логику для изделий, по аналогии с логикой по компонентам.
6. Создать логику для заказов. В логике должны быть метод получения списка заказов, метод создания заказа, метод перевода заказа в работу, метод перевода заказа в «Готово» и метод перевода заказа в «Выдан».
7. Создать реализацию модели сущности «Заказ».
8. Создать реализацию интерфейсов IProductStorage и IOrderStorage в проекте хранилища данных.
9. Создать форму и логику для списка изделий по аналогии с формой и логикой для списка компонент.
10. В форме FormCreateOrder дописать логику загрузки списка изделий в выпадающий список.
11. В форме FormMain дописать логику вывода списка заказов на форму и вызова формы со списком изделий.

Порядок сдачи базовой части

1. Добавить компонент.
2. Добавить изделие.

3. Создать заказ.
4. Сменить статус заказа на «Выполняется».
5. Сменить статус заказа на «Готов».
6. Сменить статус заказа на «Выдан».
7. Ответить на вопрос преподавателя.

Контрольные вопросы к базовой части

1. Как подключать свои и сторонние библиотеки к проекту?
2. Какова архитектура проекта, какие слои содержит, за что отвечает каждый слой?
3. Как работает IoC-контейнер?

Усложненная лабораторная (необязательно)

1. Создать сущность «Магазин» (не должно быть 2-х магазинов с одним и тем же названием) поля: название, адрес, дата открытия.
2. Предусмотреть возможность размещения любого изделия в любом магазине в любом количестве.
3. В логике работы по магазинам реализовать методы поставки изделий в определенном количество (при этом, если такое изделие уже есть в магазине, то увеличить его количество).
4. В реализации интерфейса магазина предусмотреть вывод магазина со списком изделий, находящихся в нем, но не делать в логике добавления и редактирования магазина работу с изделиями (т.е. получение списка и элемента как у «Изделия», а добавление и редактирование как у «Компонента», удаление как у «Изделия»).
5. Создать форму для отображения списка магазинов (аналогичную как для «Компонента» или «Изделия»).
6. Создать форму для работы с магазином (ввод данных по магазину и отображение списка изделий в этом магазине, но без возможности добавления/редактирования/удаления изделий).

7. На главной форме в меню в пункте «Справочники» добавить пункт «Магазины» и вызывать через него форму со списком магазинов.
8. Создать форму для поступления изделий в магазин (на форме предусмотреть возможность выбора магазина, выбора изделия и ввода количества единиц пополнения).
9. На главной форме в меню добавить пункт «Пополнение магазина» и вызывать форму пополнения магазина при выборе этого пункта.

Порядок сдачи усложненной части

1. Создать компонент.
2. Создать изделие
3. Создать магазин.
4. Пополнить магазин изделием на произвольное количество.
5. Показать магазин, в котором отображается добавленное изделие с введенным количеством.
6. Повторно пополнить магазин этим изделием.
7. Показать магазин, в котором отображается изделие с правильным значением количества.
8. Ответить на вопрос преподавателя.

Контрольные вопросы к усложненной части

1. Какова логика пополнения магазина?
2. Как сущность «Магазин» встроена в проект?
3. Какова логика работы формы создания/редактирования магазина?

Варианты

1. Кондитерская. В качестве компонентов выступают различные виды шоколада и наполнители, типа орехов, изюма и т.п. Изделие – кондитерское изделие (pastry).

2. Автомастерская. В качестве компонентов выступают различные масла, смазки и т.п. Изделия – ремонт автомобиля (repair).
3. Моторный завод. В качестве компонентов выступают различные детали для производства двигателей. Изделия – двигатели (engine).
4. Суши-бар. В качестве компонентов выступают различные продукты для суши (рыба, водоросли, соусы). Изделия – суши (sushi).
5. Продажа компьютеров. В качестве компонентов выступают различные части для компьютеров (планки памяти, жесткие диски и т.п.). Изделия – компьютеры (computer).
6. Сборка мебели. В качестве компонентов выступают различные заготовки (ножки, спинки и т.п.). Изделия – мебель (furniture).
7. Рыбный завод. В качестве компонентов выступают различные виды рыб + дополнения к ним, типа соусов и т.п. Изделия – консервы (canned).
8. Установка ПО. В качестве компонентов выступают различное ПО. Изделия – пакеты установки, например, пакет установки офисных приложений, пакет разработчика и т.п. (package).
9. Ремонтные работы в помещении. В качестве компонентов выступают различные расходные материалы (клей, обои, краска, плитка, цемент и т.п.). Изделия – ремонтные работы в различных помещениях (repair).
10. Кузнечная мастерская. В качестве компонентов выступают различные болванки (заготовки), из которых изготавливаются подковы, кочерги и т.п. Изделия – кузнечные изделия (manufacture).
11. Пиццерия. В качестве компонентов выступают различные ингредиенты для пицц (тесто, соусы, паста и т.д.). Изделия – пиццы (pizza).
12. Завод ЖБИ. В качестве компонентов выступают различные виды бетона и металлоконструкций. Изделия – железобетонные изделия (reinforced).

13. Закусочная. В качестве компонентов выступают различные продукты для закусок (колбаса, сыр, хлеб и т.п.). Изделия – различные закуски (snack).
14. Пошив платьев. В качестве компонентов выступают различные ткани, нитки и т.п. Изделия – платья (dress).
15. Типография. В качестве компонентов выступают различные типы бумаг, тонер или чернила и т.п. Изделия – печатная продукция (листовки, брошюры, книги) (printed).
16. Автомобильный завод. В качестве компонентов выступают различные части для сборки автомобилей (кузов, двигатель, стекла и т.п.). Изделия – автомобили (car).
17. Юридическая фирма. В качестве компонентов выступают различные бланки для документов. Изделия – пакеты документов, например, для страховки или завещания (document).
18. Туристическая фирма. В качестве компонентов выступают различные условия поездки (отель проживания, туры в рамках поездок). Изделия – туристические путевки (travel).
19. Цветочная лавка. В качестве компонентов выступают различные цветы и украшения к ним. Изделия – цветочные композиции (flower).
20. Ювелирная лавка. В качестве компонентов выступают различные драгоценные камни и металлы. Изделия – драгоценности (jewel).
21. Авиастроительный завод. В качестве компонентов выступают различные части для сборки самолета (двигатели, крылья, фюзеляж и т.п.). Изделия – самолеты (plane).
22. Магазин подарков. В качестве компонентов выступают различные упаковочные материалы, ленты и подарки. Изделия – подарочные наборы (gift).
23. Система безопасности. В качестве компонентов выступают различные камеры, датчики и т.п. Изделия – базовые комплектации охраны, продвинутые, для предприятий, для частных и т.п. (secure).

24. Заказы еды. В качестве компонентов выступают различные блюда. Изделия – это наборы блюд (типа обеденный набор, или утренний набор, или набор для пикника) (dish).
25. Ремонт сантехники. В качестве компонентов выступают различные трубы, прокладки, смесители т.п. Изделия – замены смесителей, труб и т.п. (work).
26. Лавка с мороженым. В качестве компонентов выступают различные виды мороженного и добавки (орехи, шоколад и т.п.). Изделия – мороженное (icecream).
27. Судостроительный завод. В качестве компонентов выступают различные части для сборки судов (корпуса, двигатели и т.п.). Изделия – суда (ship).
28. Столярная мастерская. В качестве компонентов выступают различные деревянные заготовки. Изделия – деревянные игрушки, утварь и т.п. (wood).
29. Бар. В качестве компонентов выступают различные ингредиенты для коктейлей. Изделия – коктейли (cocktail).
30. Швейная фабрика. В качестве компонентов выступают различные заготовки для штор, покрывал и т.п. (textile).