Компания «Информационно-вычислительные Системы»

Технологический отдел

Лабораторные работы по CASEBERRY

№ 3, 4

Раздел: Обучение

ТЕХНОЛОГИЧЕСКИЙ ОТДЕЛ

Лабораторные работы по CASEBERRY

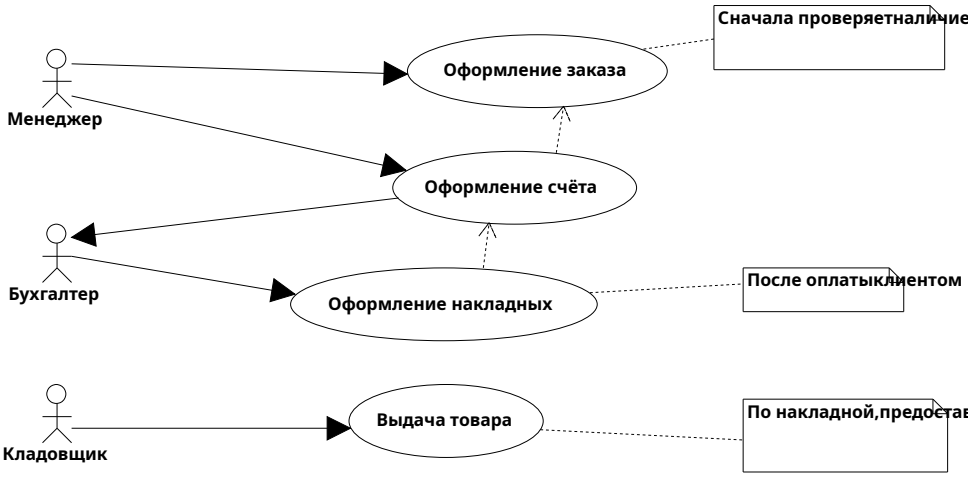
Компания ИВС
г Пермь 614007 ул. Островского 65
Тел.: (3422) 385-207, 196-500 Факс: (3422) 196-510

Лабораторная работа №3 ..... *4*
Создание отдельных приложений пользователей **4**
Создание приложения для бухгалтера **7**
Создание приложения для кладовщика **8**
Настройка кнопок на списковых формах **10**
Настройка панели инструментов формы редактирования **12**
Представления **13**
Исправление диаграммы классов – добавление поля «Должность» для класса «Сотрудник» **15**
Контрольные вопросы **15**
Лабораторная работа №4 ..... *16*
Заполнение данными для проведения тестов **16**
Наложение ограничения на список сотрудников для выбора сотрудника, заполнившего заявку **17**
Указание даты формирования заказа **19**
Автоматическое вычисление цены товара в заказе **19**
Наложение ограничения на список выбираемых товаров в заказе: **23**
Проверка количества товаров, указанного менеджером в заказе **24**
Автоматическое получение списка товаров в накладной **25**
Организация проверки наличия товара на складе при оформлении заказа **26**
Понятие бизнес-сервера 26
Блокирование заказа, после перехода в состояние Оплачен **30**
Контрольные вопросы **31**
Самостоятельная разработка *31*
Перспективы развития АСУ_Склад **31**

Лабораторная работа №3

Создание отдельных приложений пользователей

Займёмся доработкой проектируемой системы в соответствии с требованиями. Сначала вернёмся к диаграмме прецедентов, и посмотрим, какие акторы должны вступать во взаимодействие с будущей системой.

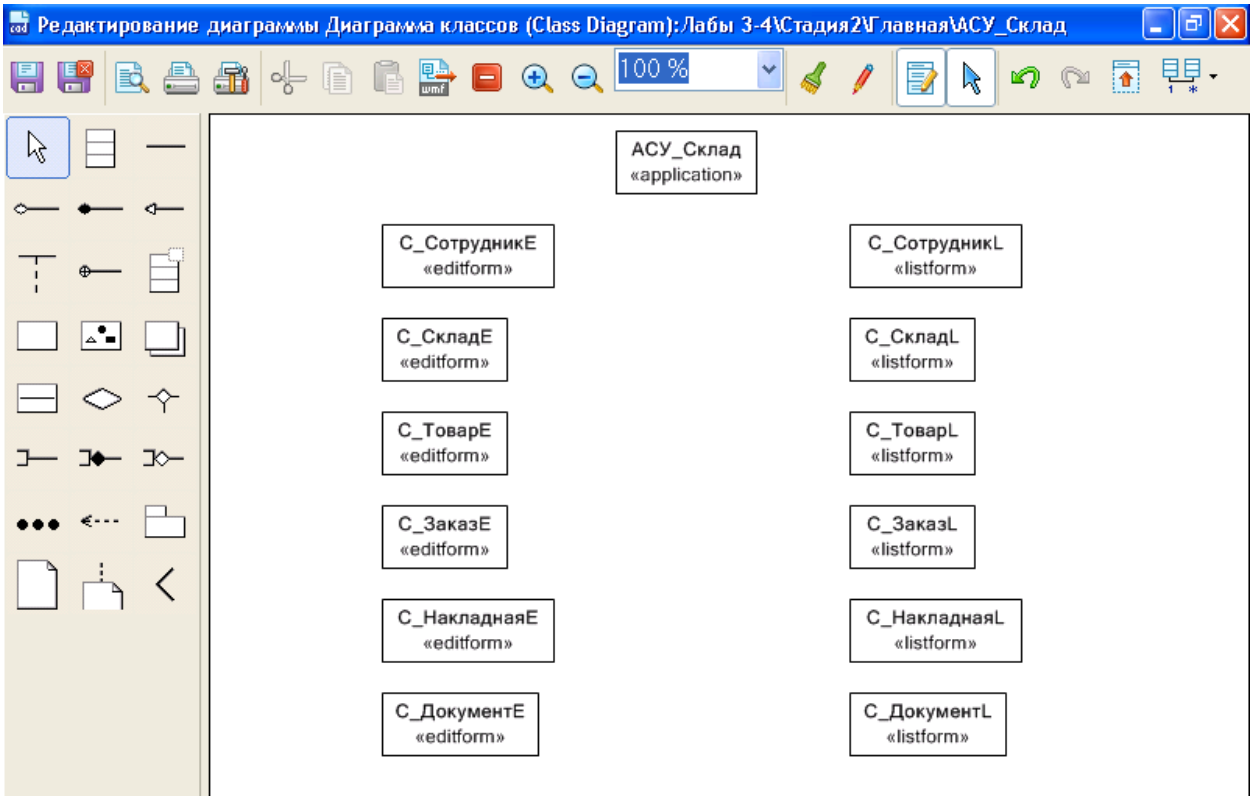


- Менеджер должен иметь возможность работать с товарами, заказами, складами и, пусть, со списком сотрудников. (Исправим в соответствии с этим новым требованием диаграмму прецедентов).
- Бухгалтер должен работать с накладными.
- Кладовщик должен иметь доступ к списку товаров и накладных.

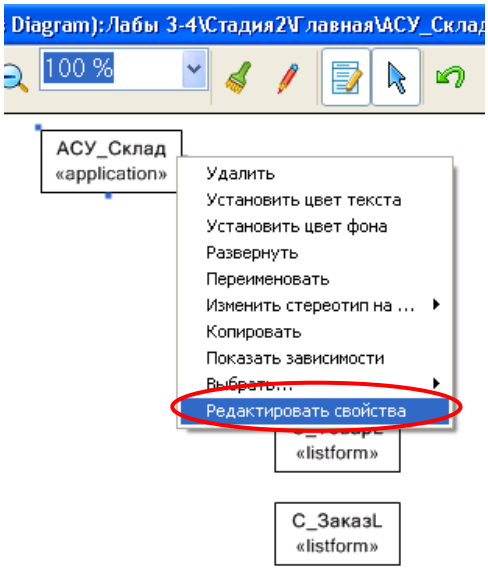
В соответствии с определёнными выше правилами создадим классы со стереотипом «application».

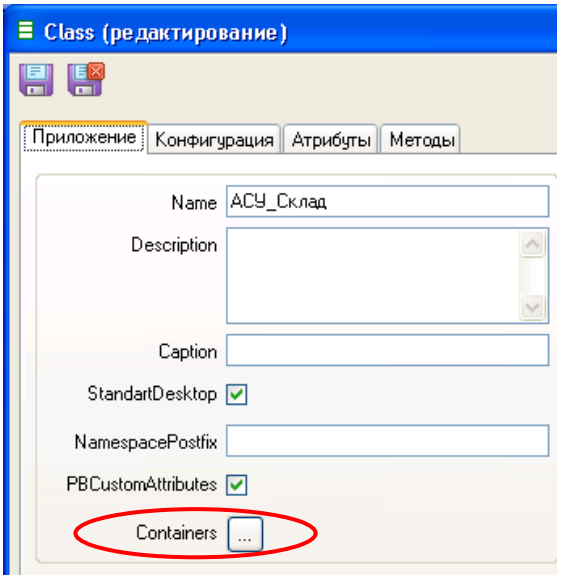
Настроим приложение для менеджера:

- Откроем диаграмму, полученную автоматически с помощью быстрого прототипирования (АСУ_Склад):

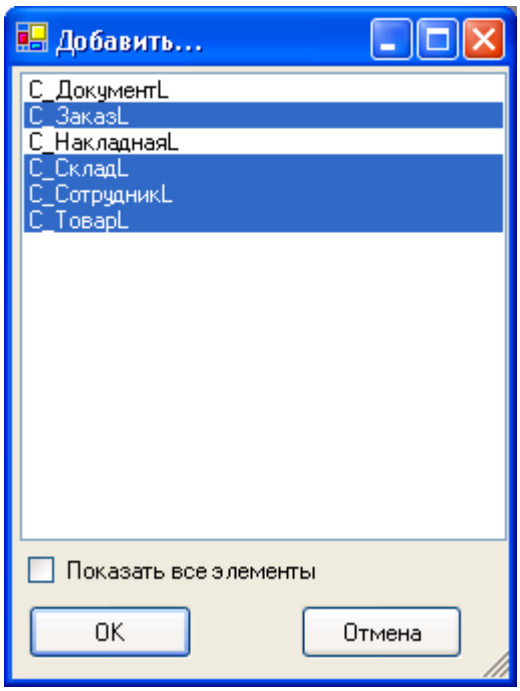


- Настроим приложение АСУ_Склад в соответствии с требованиями, предъявляемыми к приложению для менеджера. Для этого сначала откроем окно с настройками:

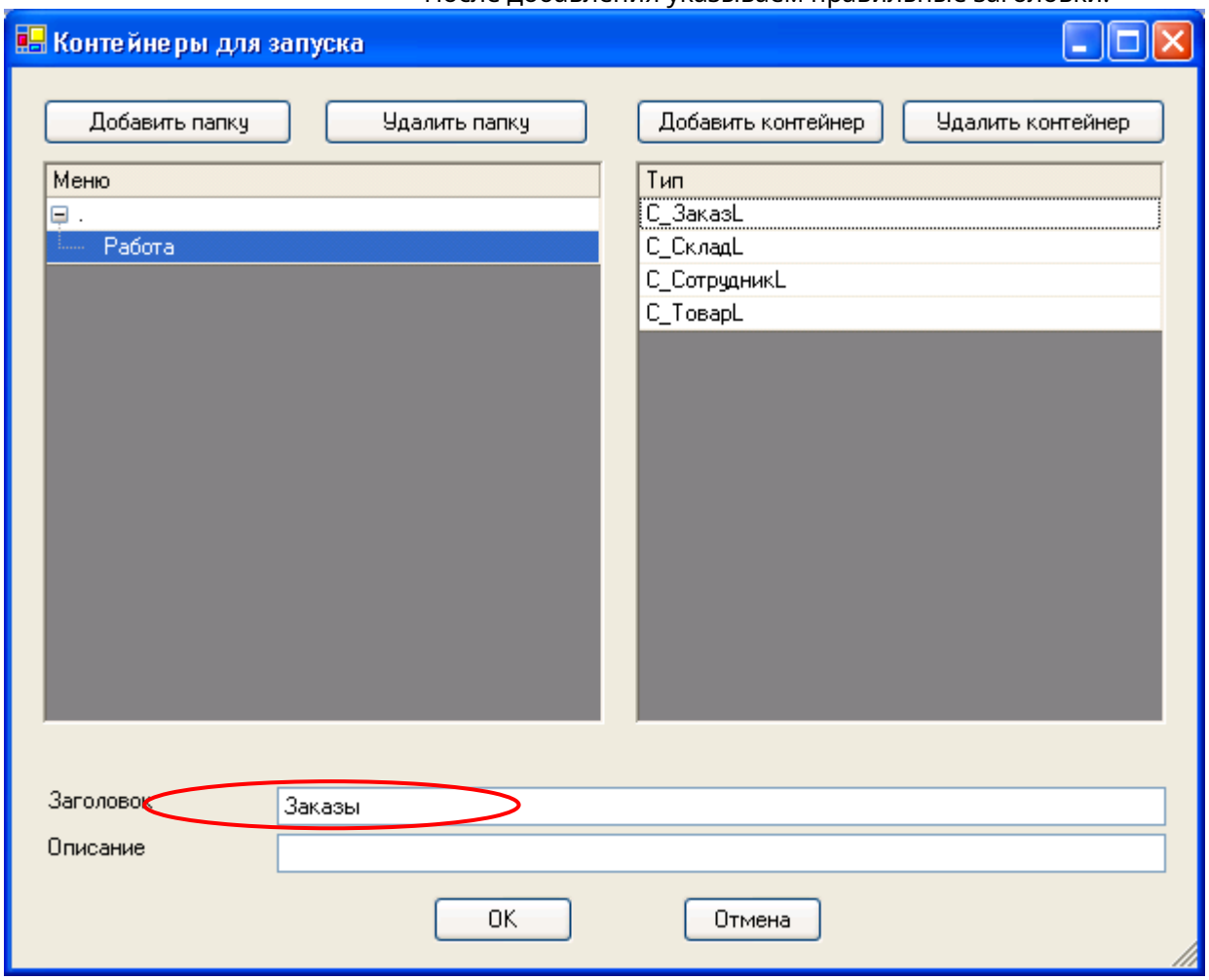




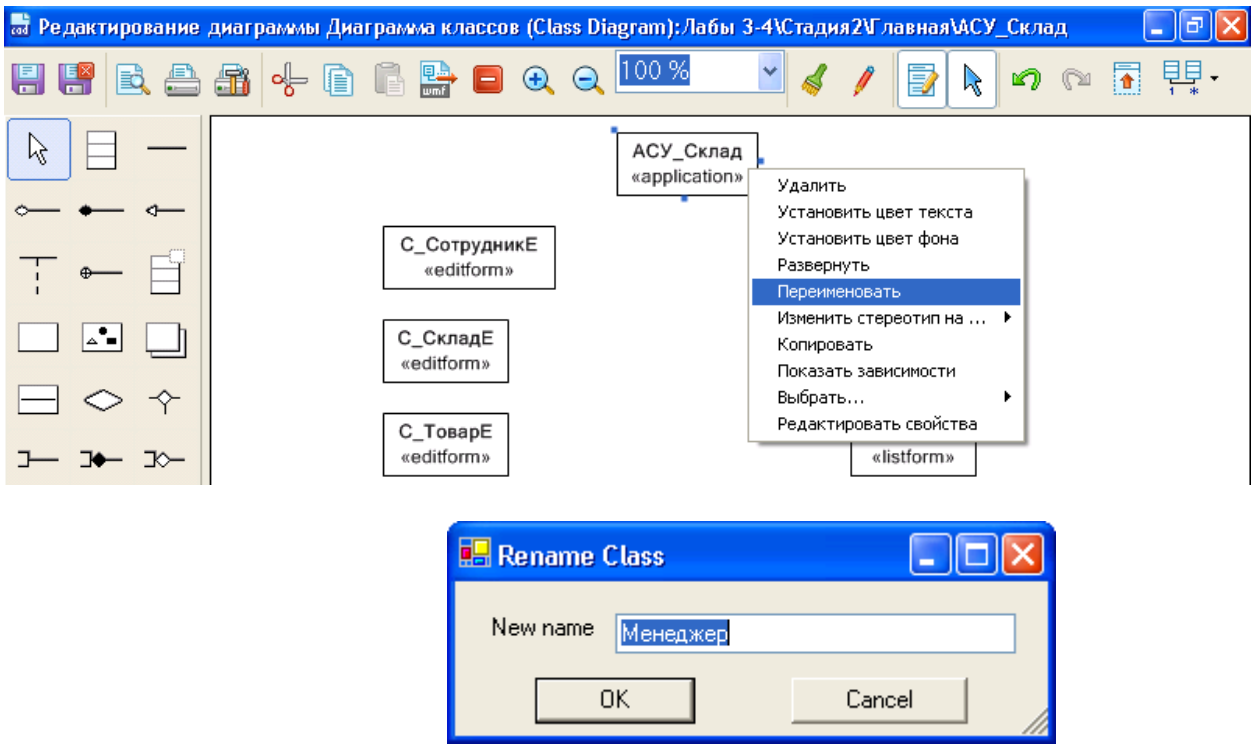
- В окне добавления форм указываем все необходимые контейнеры для запуска:



- После добавления указываем правильные заголовки:

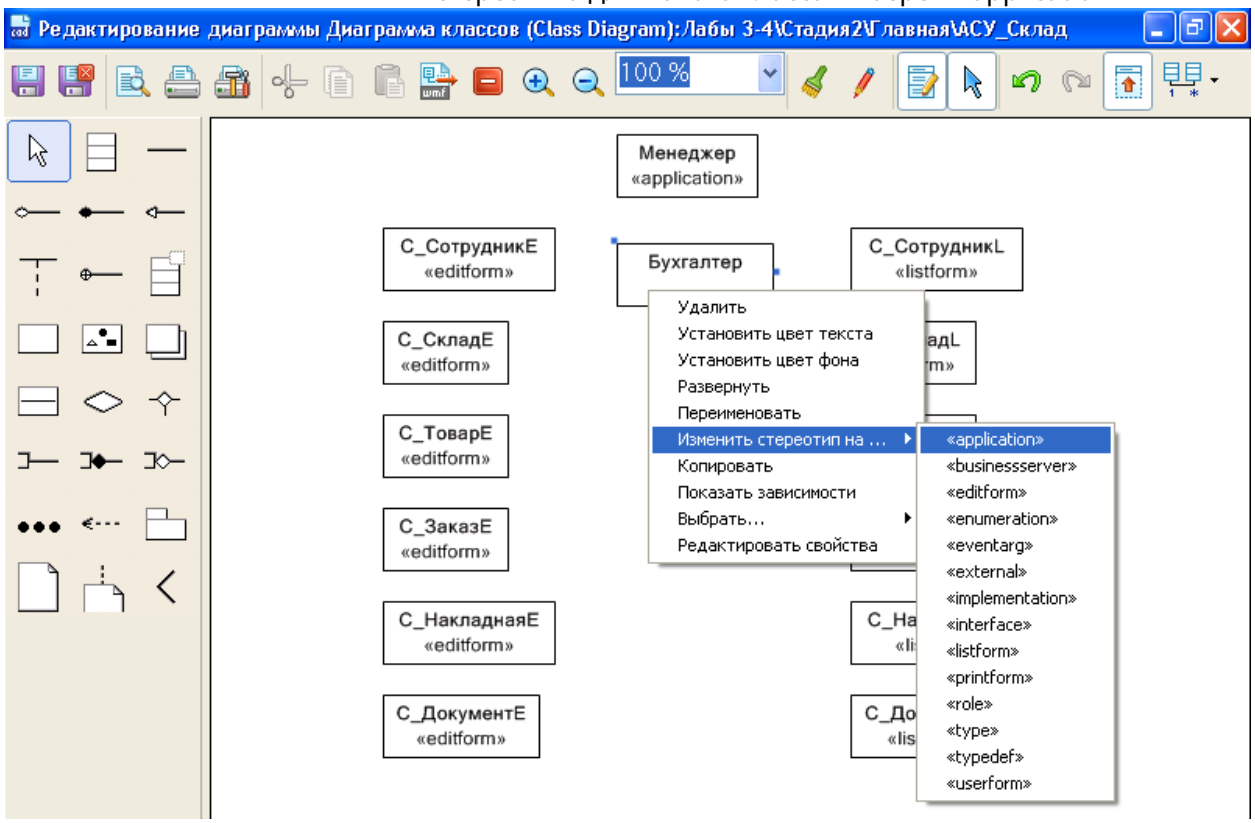


- В конце сохраняем настройки контейнеров запуска, нажав на «ОК» и закрываем с сохранением окно редактирования свойств класса АСУ_Склад.
- Переименуем класс «АСУ_Склад» в «Менеджер»

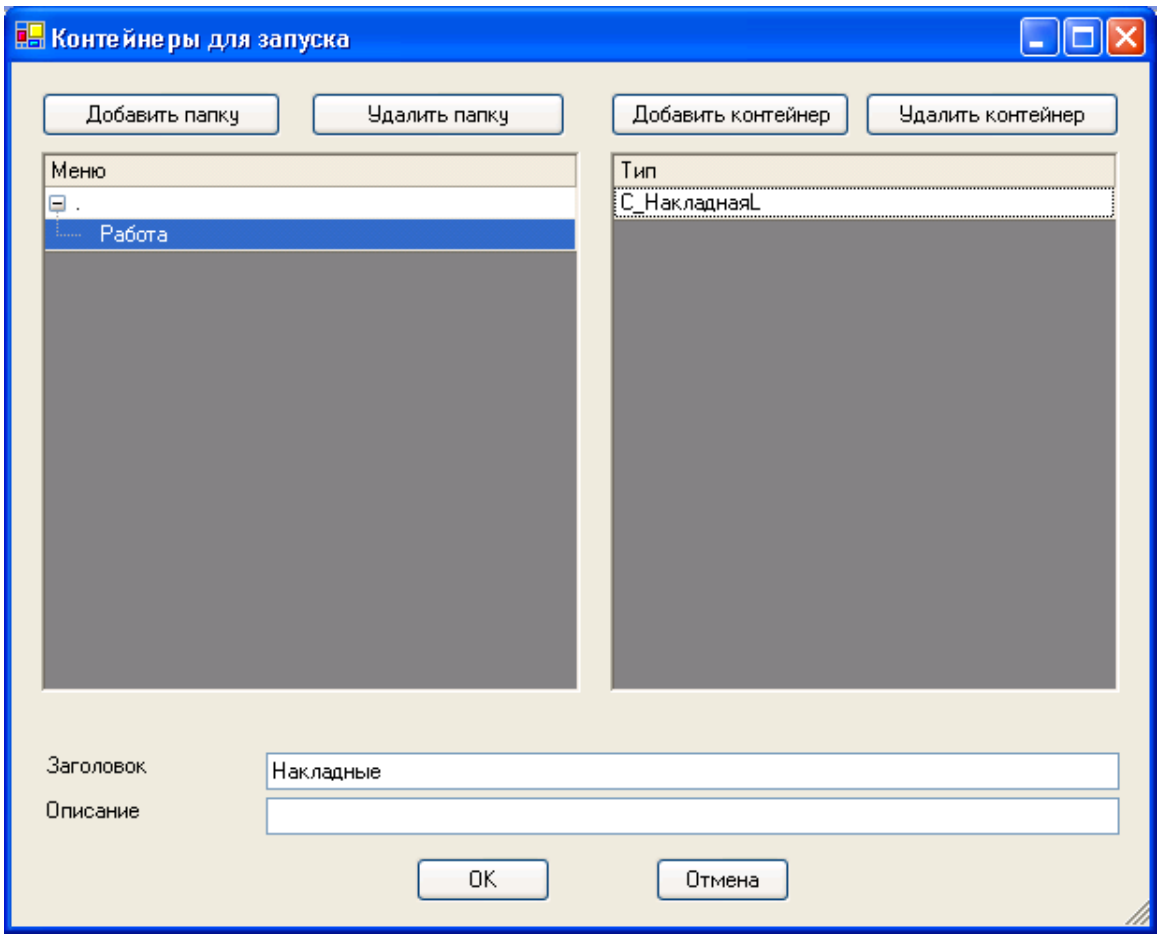


Создание приложения для бухгалтера

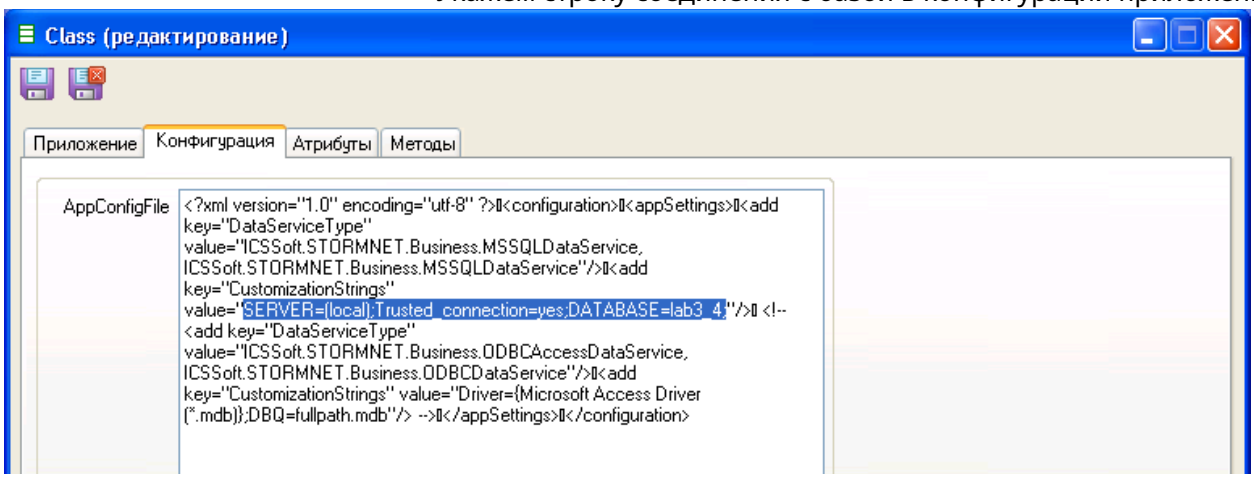
- Нарисуем новый класс и назовём его Бухгалтер. В качестве стереотипа для нового класса выберем «application»



- Откроем окно с настройками контейнеров запуска для приложения «Бухгалтер» и настроим, как показано ниже:



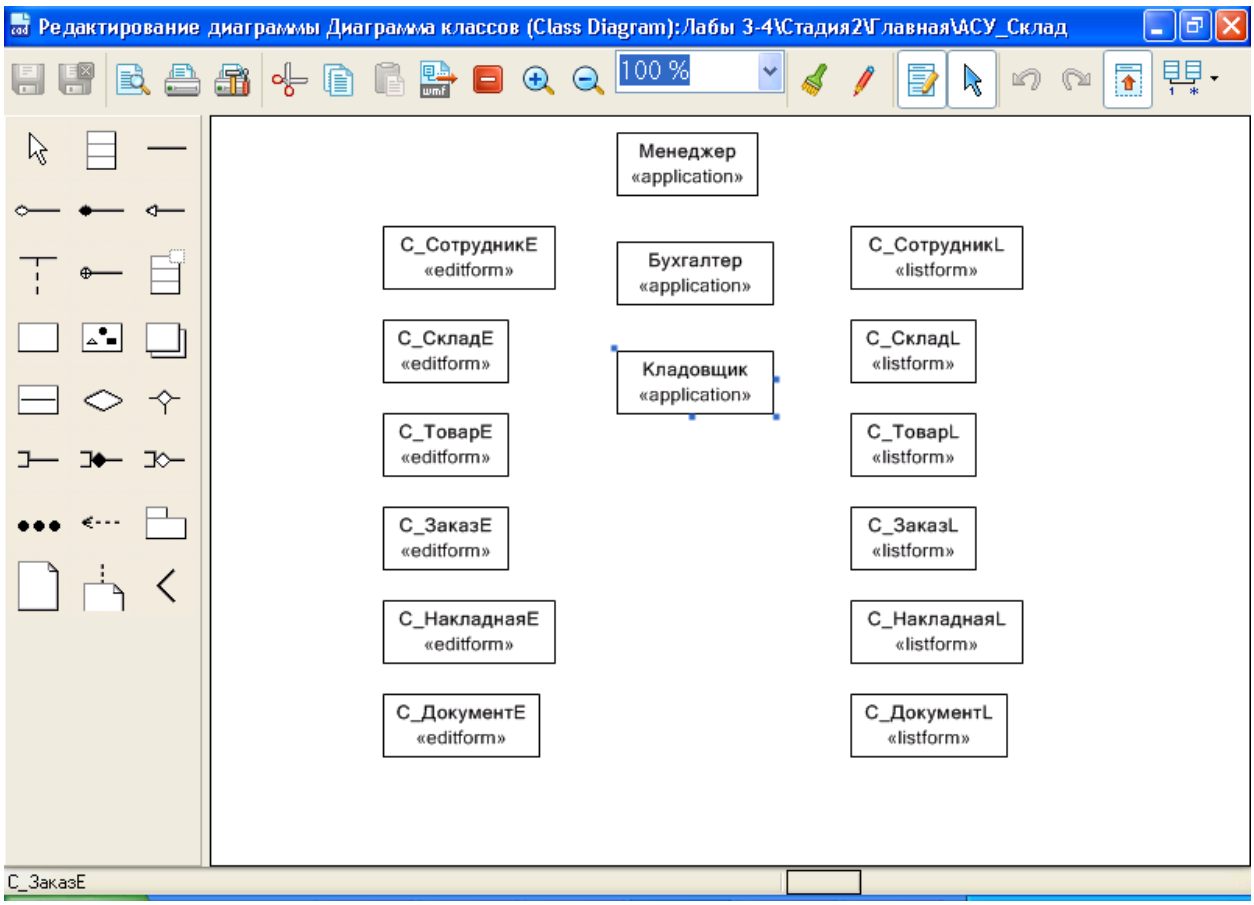
- Сохраняем настройки;
- Укажем строку соединения с базой в конфигурации приложения:



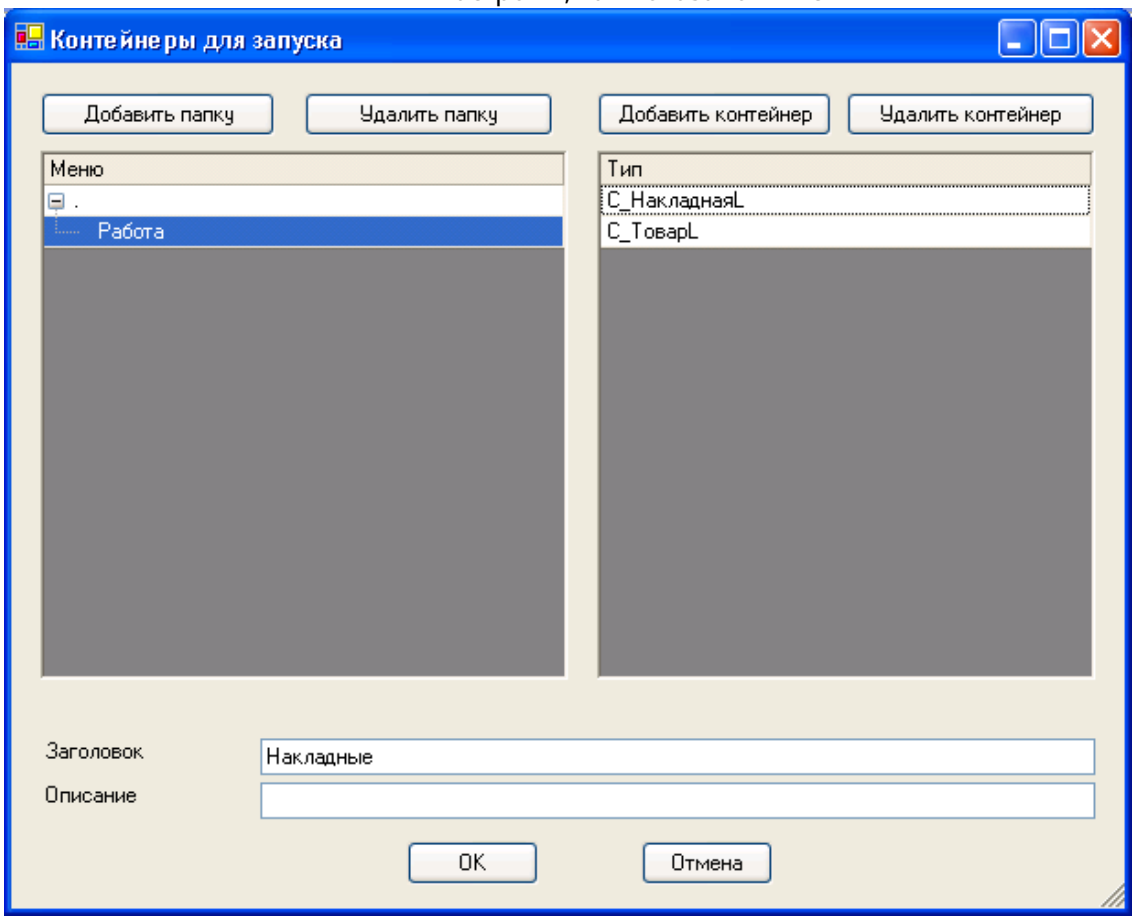
- Сохраним всё.

Создание приложения для кладовщика

- Нарисуем новый класс



- Настроим, как показано ниже

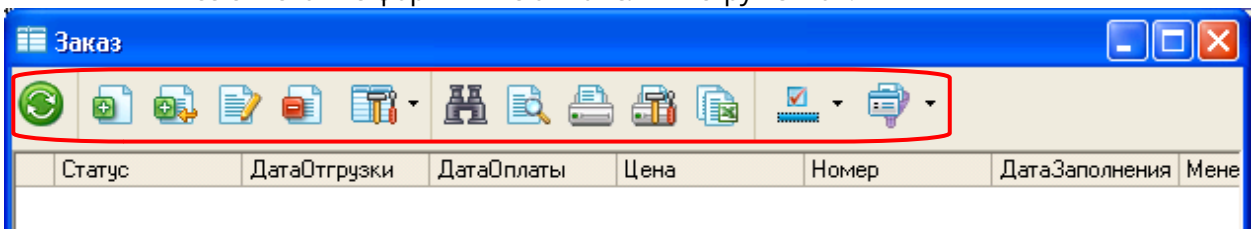


- Укажем строку соединения с базой (смотри, как это делалось с приложением «Бухгалтер»)

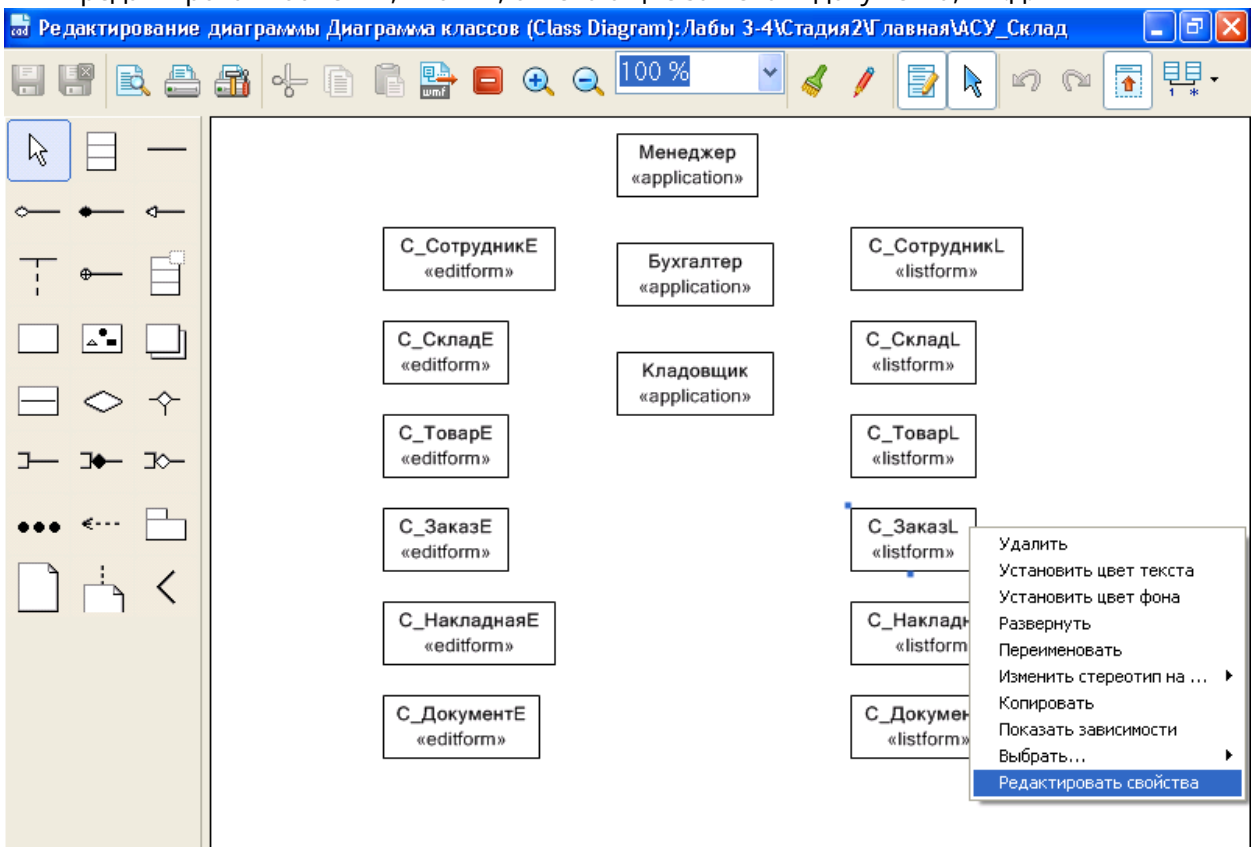
- Сохраним.

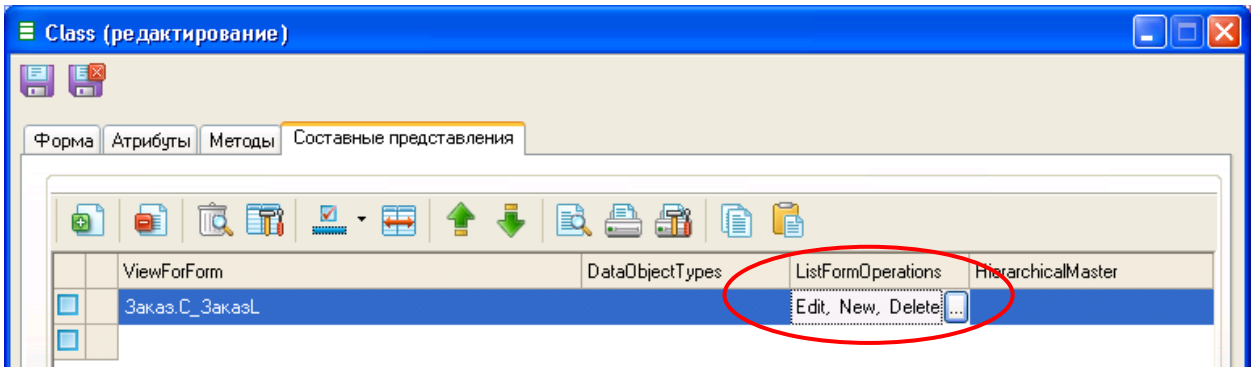
Настройка кнопок на списковых формах

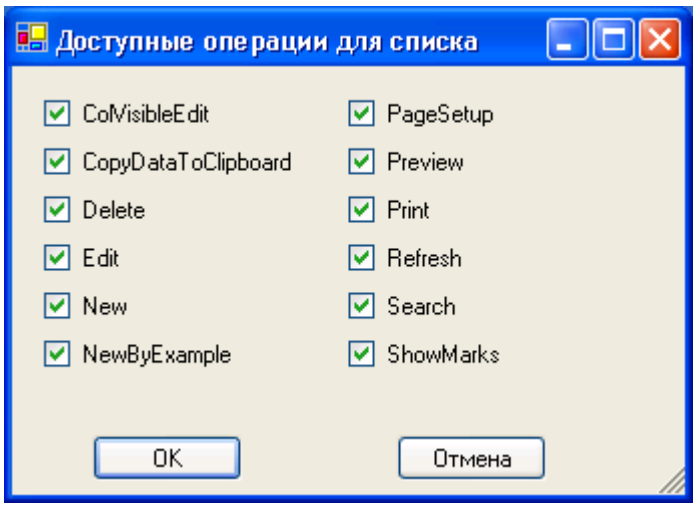
Все списковые формы имеют панель инструментов:



Эту панель можно настраивать. Например, можно убрать кнопку, позволяющую редактировать объекты, кнопки, отвечающие за печать документа, и т.д.

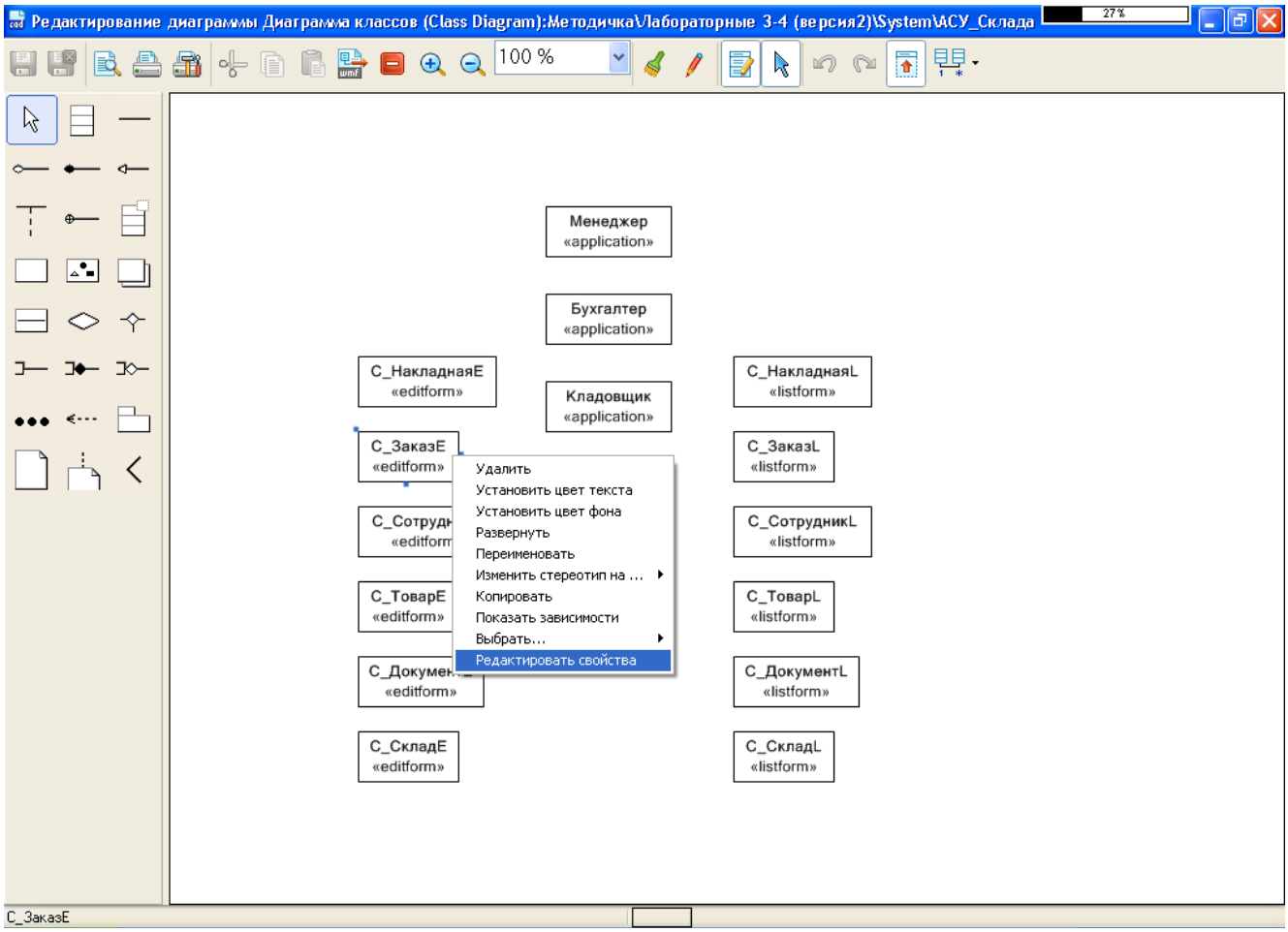


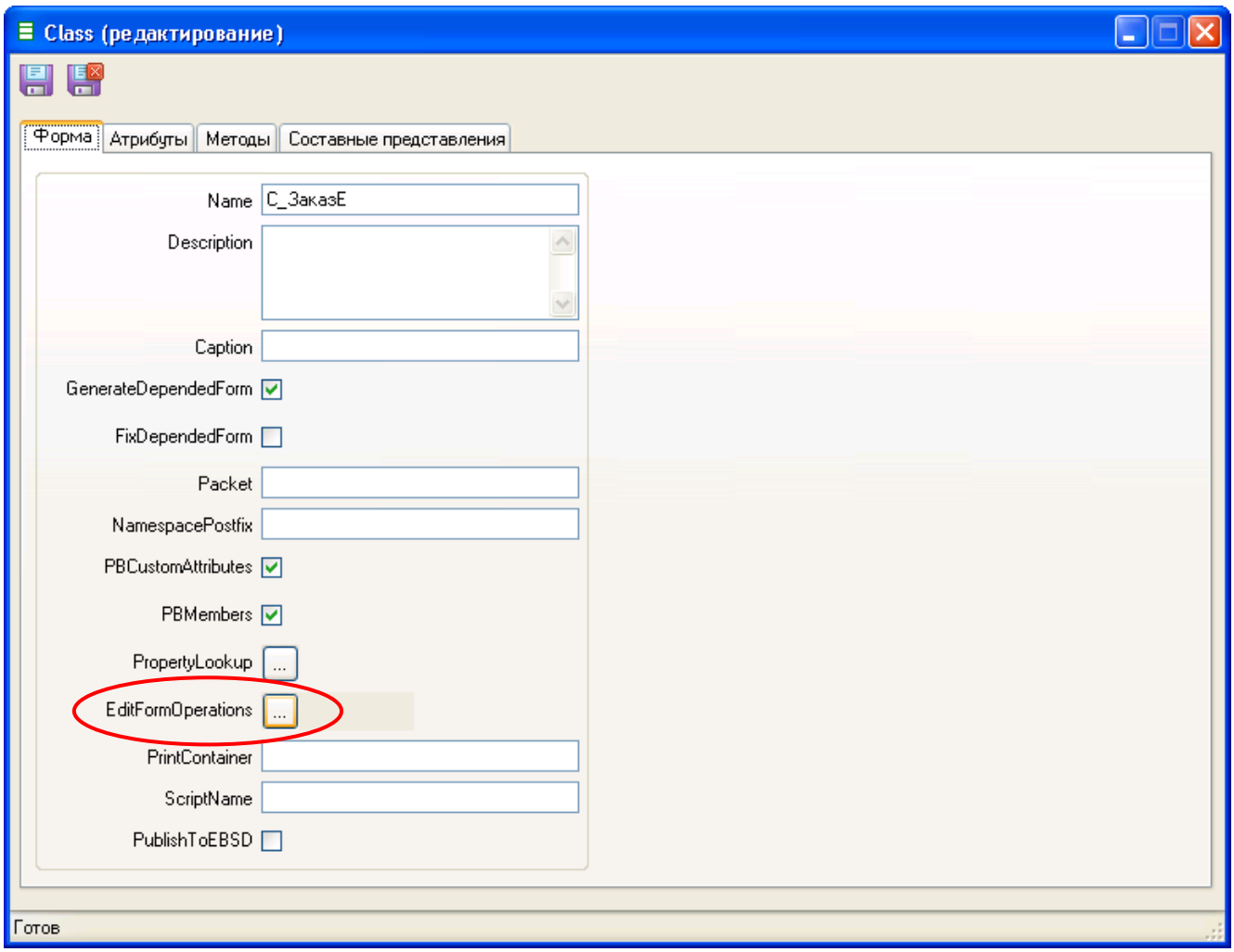




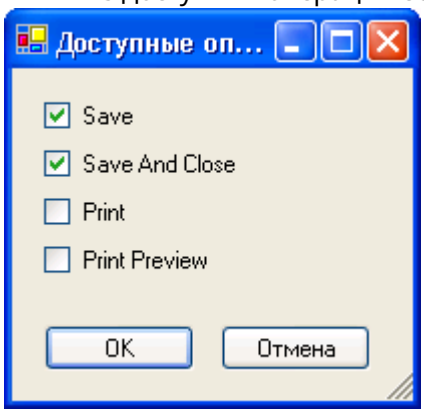
На этой форме можно отредактировать операции, которые доступны для списка. Изменения вступят в силу после перегенерации-перекомпиляции форм.

Настройка панели инструментов формы редактирования





Из доступных операций есть только сохранение и печать:



Представления

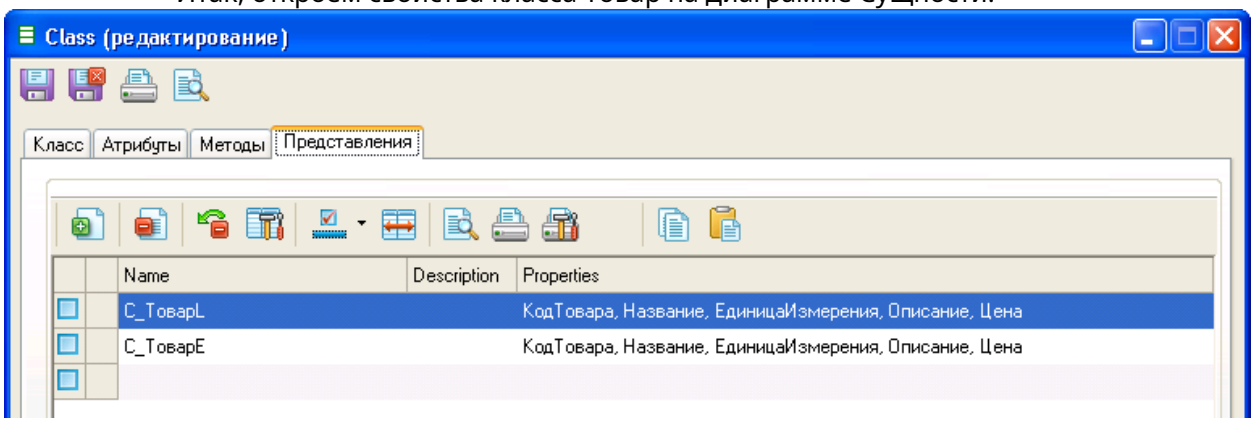
Представление необходимо для того, чтобы указать, какие именно атрибуты видны на формах и как они называются на формах, а какие — нет[1].

Рассмотрим, как можно изменить состав представления и названия на формах.

[1] В действительности понятие представления значительно шире — это некоторое подмножество всех атрибутов, включая атрибуты связанных классов. В общем, смысл представления проистекает из простого вопроса: допустим, необходимо прочитать некоторый объект из базы данных: какие атрибуты должны быть прочитаны? Ну, не все же подряд читать. Аналогичные задачи указания множества атрибуты возникают и во многих пользовательском интерфейсе.

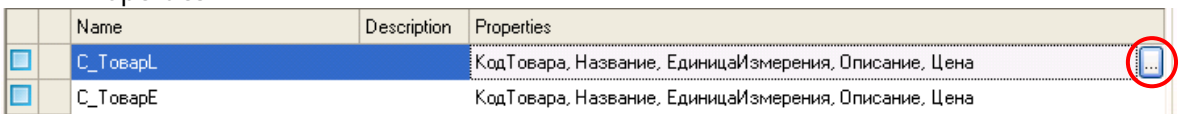
Чтобы поля в списковых формах и на формах редактирования назывались красиво (например, «Код товара», а не «КодТовара»), нам нужно исправить соответствующие значения в представлениях классов объектов данных.

Итак, откроем свойства класса Товар на диаграмме Сущности:

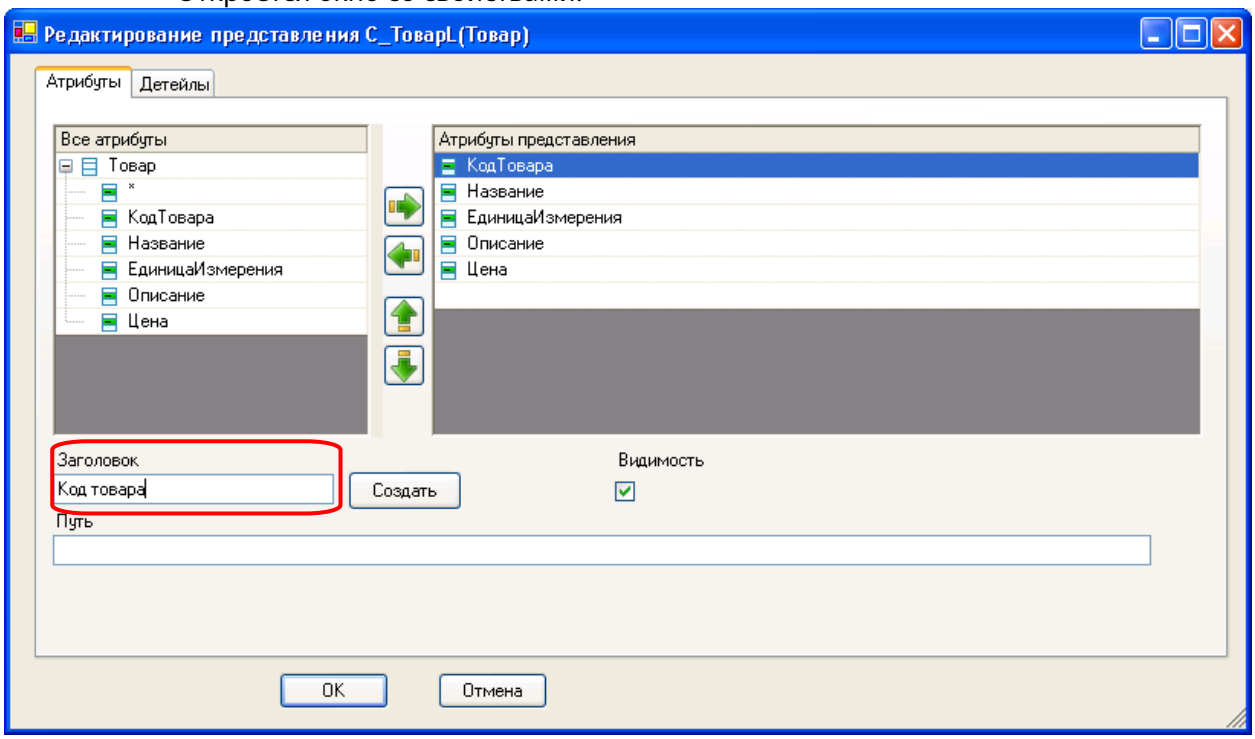


Здесь мы видим два представления, созданные автоматически при быстром прототипировании. Первое представление оканчивается на «L», это значит, что оно будет применяться для отображения объектов на списковой форме (listform). Второе представление имеет на конце имени букву «E», что говорит о применении данного представления при создании формы редактирования (editform). Если в приложении используются печатные формы, то представления для них рекомендуется называть так, чтобы на конце имени была буква «P» (printform).

Для того чтобы отредактировать представление, нажмём на кнопку «...» в поле Properties



Откроется окно со свойствами:

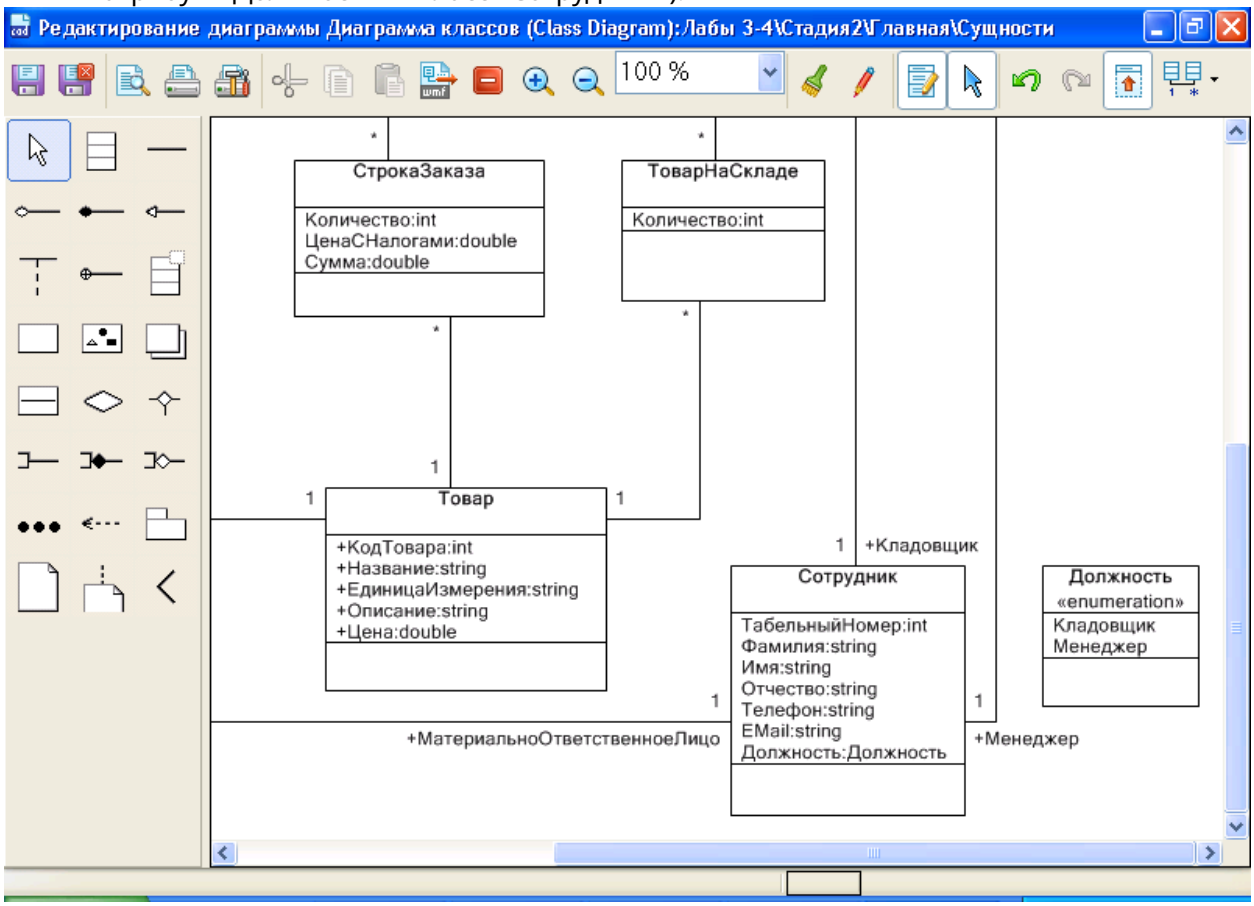


В этом окне можно выбрать необходимые атрибуты и для каждого атрибута задать заголовок. Этот заголовок и будет отображаться на списковой форме или форме редактирования.

В соответствии с приведённым примером исправьте заголовки атрибутов для всех представлений всех классов (чтобы слова были раздельно, а с заглавной буквы было бы только первое слово).

Исправление диаграммы классов – добавление поля «Должность» в класс «Сотрудник»

В дальнейшем нам нужно будет накладывать ограничение на список сотрудников, чтобы при заполнении соответствующих документов нельзя было выбрать менеджера вместо кладовщика. Внесём соответствующие изменения (вводится перечислимый тип «Должность» и атрибут «Должность» в класс «Сотрудник»):



Также нужно изменить представления *C_СотрудникL* и *C_СотрудникE* так, чтобы атрибут «Должность» попал в них.

Перегенерируем объекты и приведём в соответствие базу данных.

Контрольные вопросы

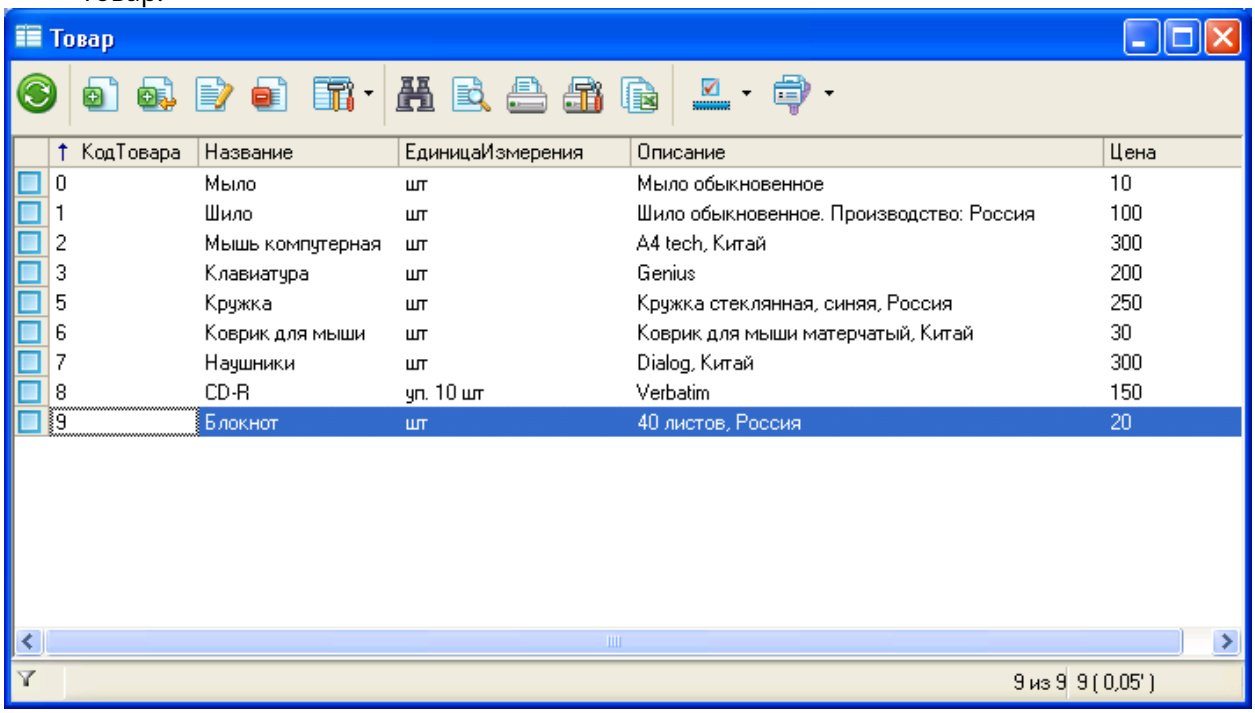
1. С какой целью созданы приложения для разных пользователей?
2. Какие основные настройки надо указать при создании класса со стереотипом «application»?
3. Что такое представление?

4. Какие шаги нужно произвести, чтобы изменения на диаграмме классов стали заметными в приложении?

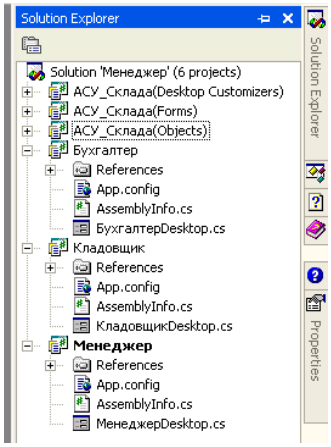
Лабораторная работа №4

Заполнение данными для проведения тестов

Чтобы проводить эксперименты с программой, заполните несколько объектов класса Товар:

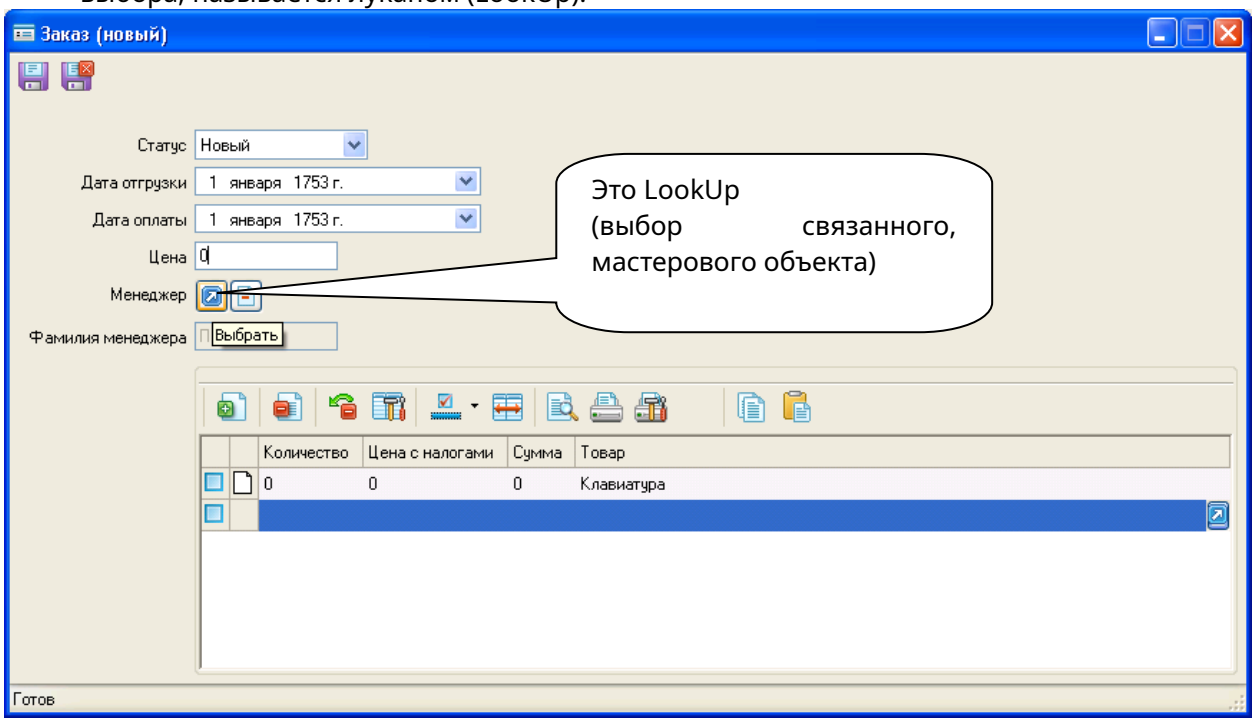


Откроем сгенерированный проект АСУ_Склада\Менеджер\Менеджер.csproj и добавим в один Solution остальные проекты для удобства программирования. Сохраним наш solution как «АСУ_Склада\Менеджер\Менеджер.sln» .

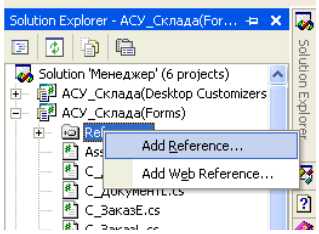


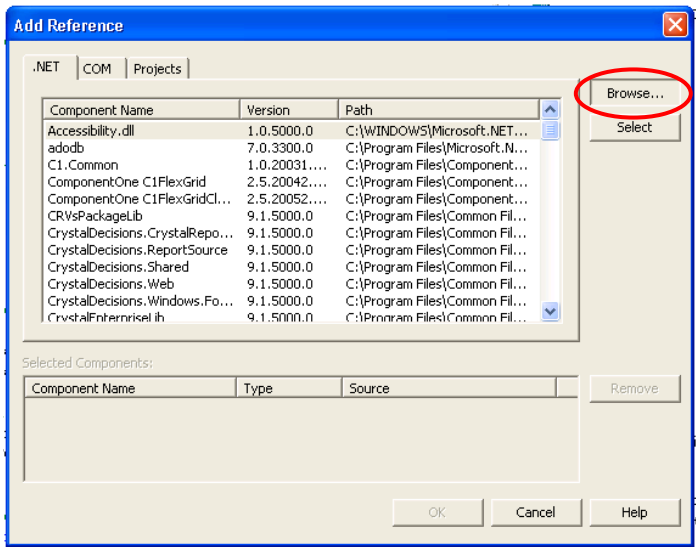
Наложение ограничения на список сотрудников для выбора сотрудника, заполнившего заказ

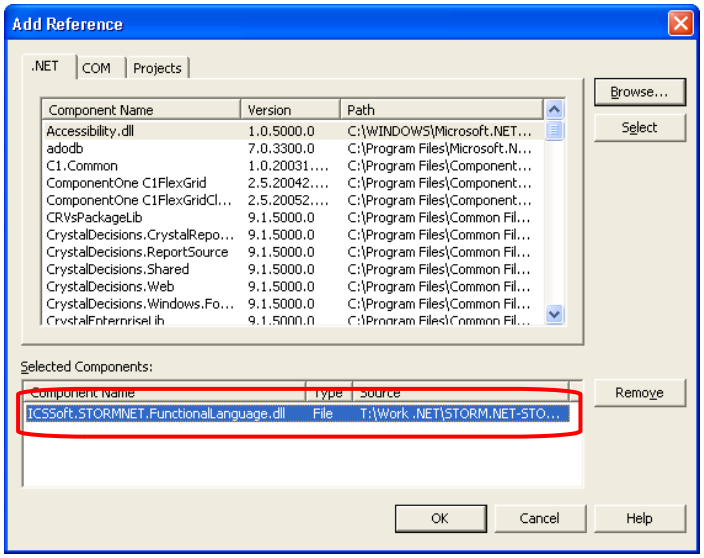
У нас в списке сотрудников находятся сотрудники с различными должностями, а нам нужно вывести только менеджеров (чтобы избежать неправильного ввода со стороны пользователя системы). В терминологии CASEBERRY кнопка, которая вызывает список для выбора, называется лукапом (LookUp).



Итак, для того, чтобы указать ограничение, откроем код независимой формы редактирования для объектов класса заказ (*C_ЗаказE.cs*). Учтём сразу то, что нам понадобится работать со сборкой ICSSoft.STORMNET.FunctionalLanguage.dll, поэтому подключим её сразу.







Выбираем соответствующую сборку из каталога с CASEBERRY.

После этого прописываем в скобках программиста следующие строчки (для выбора вариантов используется сочетание Ctrl+Пробел):

```
                //  ***  Start  programmer  edit  section  ***  (C_ЗаказE
CustomAttributes)

        using ICSSoft.STORMNET.FunctionalLanguage;
        using ICSSoft.STORMNET.FunctionalLanguage.SQLWhere;

           // *** End programmer edit section *** (C_ЗаказE CustomAttributes)
```

Далее переопределяем метод OnEdit() для нашей независимой формы (чтобы подсказчик в Visual Studio предложил выбор функций, которые можно перегружать, достаточно набрать слово override):

```
                        //  ***  Start  programmer  edit  section  ***  (C_ЗаказE
CustomMembers)
      public       override       void       OnEdit(string       propertyname,
ICSSoft.STORMNET.DataObject dataobject, string contpath, object tag)
      {
        if (propertyname == "Менеджер")
        {
              SQLWhereLanguageDef langdef = SQLWhereLanguageDef.LanguageDef;
              ICSSoft.STORMNET.FunctionalLanguage.Function        lf        =
langdef.GetFunction(langdef.funcEQ,
```

```
                    new      VariableDef(langdef.StringType,      "Должность"),
ICSSoft.STORMNET.EnumCaption.GetCaptionFor(Должность.Менеджер));
                tag = lf;
            }
            base.OnEdit (propertyname, dataobject, contpath, tag);
        }

                // *** End programmer edit section *** (C_ЗаказE CustomMembers)
```

Поясним код данной функции: сначала проверяем на то, что управление к нам пришло именно от нашего лукапа, а не от какого-нибудь другого (в конкретном примере этого можно было не делать, но часто на одной форме встречается несколько лукапов). Затем, формируем специальную конструкцию Limit Function, которая позволяет накладывать ограничения.

Важно помнить, что добавлять свой код необходимо только в скобках программиста (между словами `Start programmer edit section` и `End programmer edit section`), иначе при очередной перегенерации исходного кода, внесённые изменения будут потеряны.

Перекомпилируем Solution, запустим систему и проверим, как работают внесённые изменения (при нажатии на кнопку лукапа открывается список, в котором только менеджеры).

Указание даты формирования заказа

Все документы должны иметь дату заполнения, поэтому нам надо предусмотреть и соответствующую обработку. Причём, по-умолчанию должна проставляться текущая дата. Самый простой способ добиться этого – написать на диаграмме инициализирующее значение следующим образом:

После перегенерации объектов можно будет увидеть, что ДатаЗаполнения в исходном коде класса «Документ» определена как:

```
private System.DateTime fДатаЗаполнения = System.DateTime.Now;
```

Согласно C# инициализаторы равнозначны конструктору класса по-умолчанию, поэтому при каждом конструировании нового «Документа» и любого его наследника будет проставляться текущее время.

Автоматическое вычисление цены товара в заказе

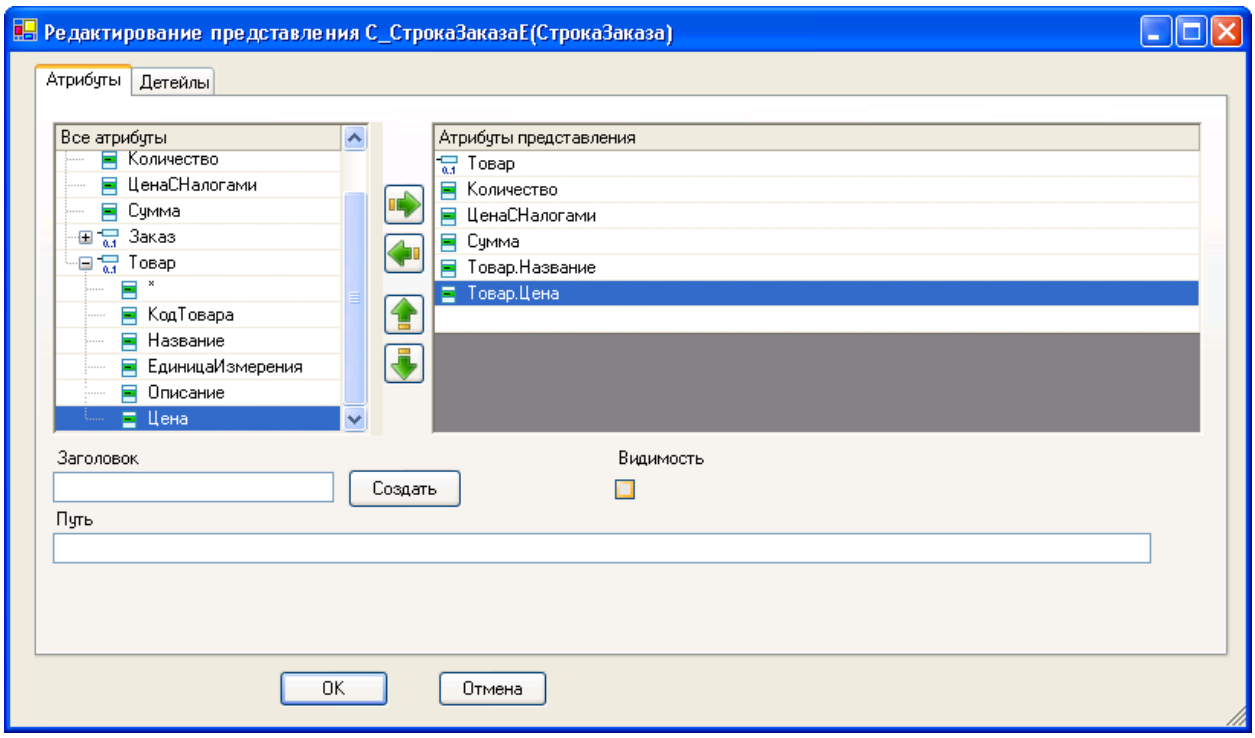
Сделаем в «СтрокеЗаказа» вычислимыми атрибуты «ЦенаСНалогами» и «Сумма», ибо их значения однозначно определяются значением «Цены» «Товара» и «Количества» в «СтрокеЗаказа»:

В *СтрокаЗаказа.cs* в коде аксессоров Get для свойств «ЦенаСНалогами» и «Сумма» пропишем вычисления атрибутов.

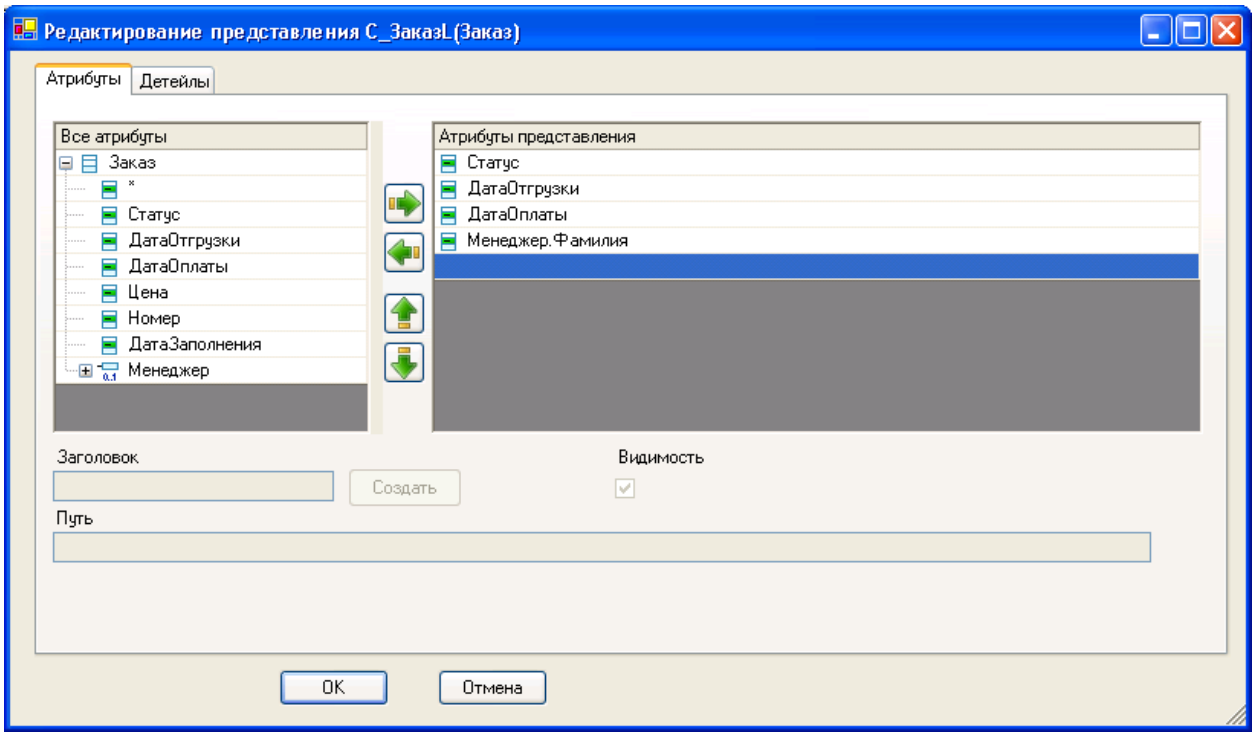
```
            [NotStored()]
            public virtual System.Double ЦенаСНалогами
```

```
            {
                get
                {
                                    // *** Start programmer edit section ***
(СтрокаЗаказа.ЦенаСНалогами Get)
                        Double ценасналогами = 0;
                        if ( Товар != null && Товар.Цена > 0 )
                        {
                              ценасналогами = Товар.Цена * 1.35;
                        }
                        return ценасналогами;
                        //   ***   End   programmer   edit   section   ***
(СтрокаЗаказа.ЦенаСНалогами Get)
                }
                set
                {
                                    // *** Start programmer edit section ***
(СтрокаЗаказа.ЦенаСНалогами Set)
                                     // *** End programmer edit section ***
(СтрокаЗаказа.ЦенаСНалогами Set)
                }
            }

              // *** Start programmer edit section *** (СтрокаЗаказа.Сумма
CustomAttributes)

             // *** End programmer edit section *** (СтрокаЗаказа.Сумма
CustomAttributes)
            [NotStored()]
            public virtual System.Double Сумма
            {
                get
                {
                                    // *** Start programmer edit section ***
(СтрокаЗаказа.Сумма Get)
                        Double сумма = 0;
                        if ( Товар != null && Товар.Цена > 0)
                        {
                              сумма = ЦенаСНалогами * Количество;
                        }
                    return сумма;
                                     // *** End programmer edit section ***
(СтрокаЗаказа.Сумма Get)
                }
                set
                {
                                    // *** Start programmer edit section ***
(СтрокаЗаказа.Сумма Set)
                                     // *** End programmer edit section ***
(СтрокаЗаказа.Сумма Set)
                }
            }
```

Поскольку в вычислении используется Товар.Цена, необходимо, чтобы значение Товар.Цена из мастерового класса для строки заказа зачитывалось из базы, для чего надо в представление добавить это поле:

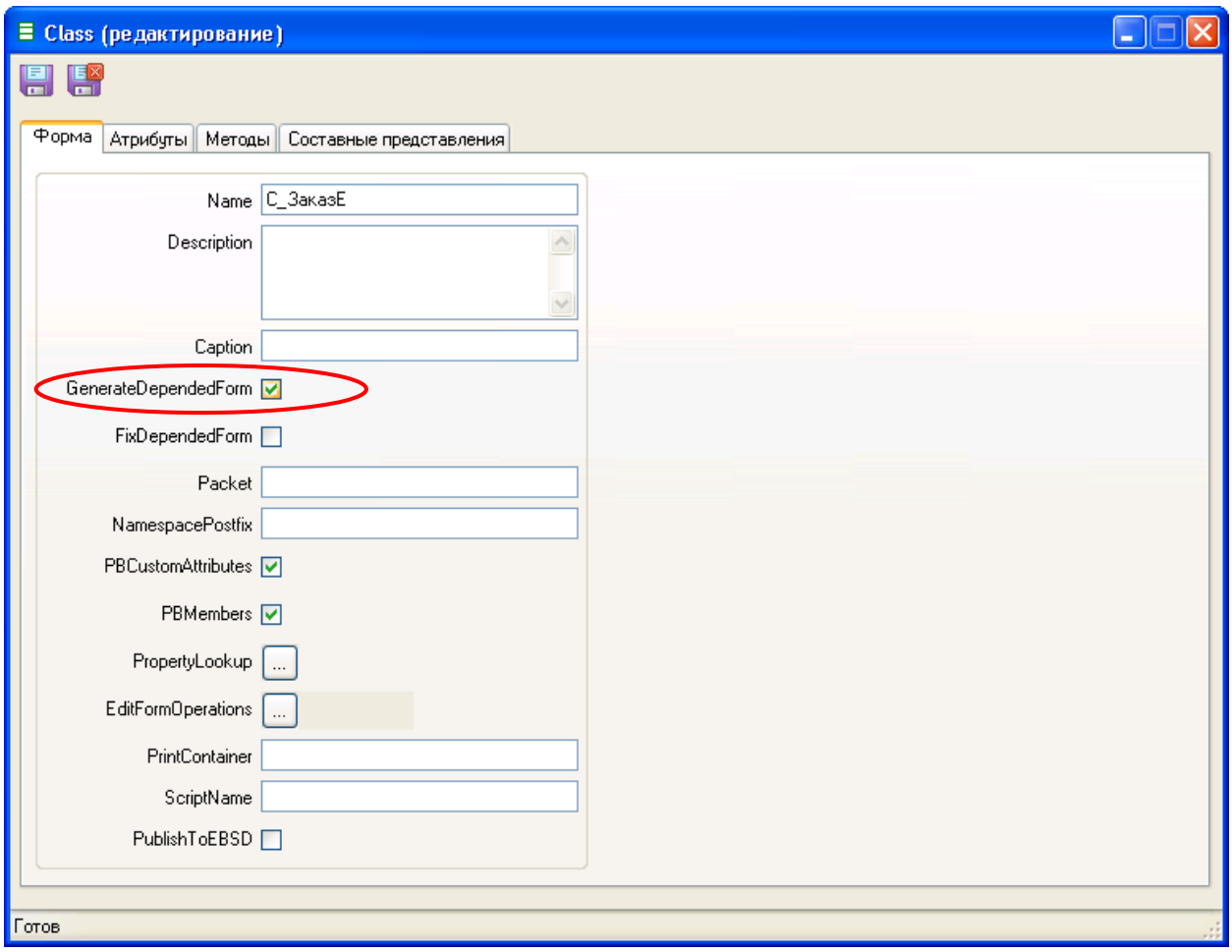


Цена больше не нужна в списке заказов: уберём поле Цена из представления для списка заказов.



После этого нужно перегенерировать-перекомпилировать объекты.

Для того, чтобы посчитать сумму всего заказа, а затем отобразить её на форме заказа, придётся форму редактирования заказа получить в виде исходного кода. Для этого поставим галочку на форме редактирования свойств класса C_ЗаказE:



И перегенерируем формы.

Откроем проект и код сгенерированной только что формы.

Для класса `WinformC_ЗаказE` перегрузим метод Edit так как показано ниже:

```
                // *** Start programmer edit section *** (WinformC_ЗаказE CustomMembers)
                public override void Edit(ICSSoft.STORMNET.DataObject dataobject, string contpath, string propertyname, object tag)
                {
                        base.Edit (dataobject, contpath, propertyname, tag);
                        EditManager.SetReadonlyFlagProperties(true, new string[] {"Цена"});
                        СтрокаЗаказа.EditManager.AfterChangeProperty -=new ICSSoft.STORMNET.Windows.Forms.Binders.AfterChangePropertyDelegate(EditManager_AfterChangeProperty);
                        СтрокаЗаказа.EditManager.AfterChangeProperty +=new ICSSoft.STORMNET.Windows.Forms.Binders.AfterChangePropertyDelegate(EditManager_AfterChangeProperty);
                }

                private void EditManager_AfterChangeProperty(object sender, ICSSoft.STORMNET.Windows.Forms.Binders.ChangePropertyEventArgs e)
                {
                        EditManager.Change("Цена");

                }
                // *** End programmer edit section *** (WinformC_ЗаказE CustomMembers)
```

Всё это сделано для того, чтобы пользователь не имел возможности руками исправить значение поля Цена. В этом методе мы привязали обработчик события

AfterChangeProperty к менеджеру редактирования[2] объекта на форме, который срабатывает после изменения какого-нибудь свойства в любом из объектов списка. А в самом методе-обработчике указываем менеджеру редактирования, что свойство изменилось и его значение необходимо обновить на форме, соответственно вызовется метод get у свойства Цена. Чтобы значение вычислялось автоматически, нужно дописать код методе get код по вычислению значения этого свойства:

```
[NotStored()]
public virtual System.Double Цена
{
    get
    {
        // *** Start programmer edit section *** (Заказ.Цена Get)
        int цена = 0;
        for (int i = 0; i < this.СтрокаЗаказа.Count; i++)
        цена += (int)this.СтрокаЗаказа[i].Сумма;
        return цена;
        // *** End programmer edit section *** (Заказ.Цена Get)
    }
    set
    {
        // *** Start programmer edit section *** (Заказ.Цена Set)

        // *** End programmer edit section *** (Заказ.Цена Set)
    }
}
```

Наложение ограничения на список выбираемых товаров в заказе:

Чтобы в заказе нельзя было указать несколько раз один товар, необходимо наложить ограничение таким образом, чтобы уже выбранные товары в списке для нового выбора более не появлялись. Для этого к ранее добавленному ограничению относительно менеджера добавляем аналогично ограничение и для товаров:

```
// *** Start programmer edit section *** (C_ЗаказE CustomMembers)
public override void OnEdit(string propertyname, ICSSoft.STORMNET.DataObject dataobject, string contpath, object tag)
{
    if (propertyname == "Менеджер")
    {
        SQLWhereLanguageDef langdef = SQLWhereLanguageDef.LanguageDef;
        ICSSoft.STORMNET.FunctionalLanguage.Function lf = langdef.GetFunction(langdef.funcEQ,
            new VariableDef(langdef.StringType, "Должность"), ICSSoft.STORMNET.EnumCaption.GetCaptionFor(Должность.Менеджер));
        tag = lf;
    }
    if (propertyname == "СтрокаЗаказа.Товар")
    {
```

[2] Менеджер редактирования (EditManager) специальный класс, ответственный за связывание свойств объектов данных и элементов пользовательского интерфейса (контролов)

```
                        СтрокаЗаказа              строказаказа              =
(СтрокаЗаказа)dataobject;
                        Заказ заказ = строказаказа.Заказ;
                        SQLWhereLanguageDef             langdef             =
SQLWhereLanguageDef.LanguageDef;

                        ICSSoft.STORMNET.FunctionalLanguage.Function   lf   =
null;

                        for (int i = 0; i < заказ.СтрокаЗаказа.Count; i++)
                        {
                              if (lf != null)
                              {
                                    if(заказ.СтрокаЗаказа[i].Товар != null)
                                          lf                                  =
langdef.GetFunction(langdef.funcAND, lf,langdef.GetFunction(langdef.funcNEQ,
                                                new
VariableDef(langdef.GuidType,                         "STORMMainObjectKey"),
заказ.СтрокаЗаказа[i].Товар.__PrimaryKey));
                              }
                              else
                              {
                                    if(заказ.СтрокаЗаказа[i].Товар != null)
                                          lf                                  =
langdef.GetFunction(langdef.funcNEQ,
                                                new
VariableDef(langdef.GuidType,                         "STORMMainObjectKey"),
заказ.СтрокаЗаказа[i].Товар.__PrimaryKey);
                              }

                        }

                        if (lf != null)
                        tag = lf;
                  }

                  base.OnEdit (propertyname, dataobject, contpath, tag);
            }

             // *** End programmer edit section *** (C_ЗаказE CustomMembers)
```

Проверка количества товаров, указанного менеджером в заказе

Мы должны предусмотреть возможные ошибки при вводе данных, чтобы пользователь не имел возможности ввести совершенно неподходящее значение. Например, пользователь при указании количества товара в заказе не должен иметь возможности вводить отрицательное значение. Для этого нужно в методе set генерировать исключение при неправильном вводе:

```
                  set
                  {
                                    // *** Start programmer edit section ***
(СтрокаЗаказа.Количество Set start)

                        if (value < 0)
                        {
                              Exception   ex   =   new   Exception("Значение
количества не может быть отрицательным");
```

```
                        throw ex;
                    }
                        // *** End programmer edit section ***
(СтрокаЗаказа.Количество Set start)
                    this.fКоличество = value;
                        // *** Start programmer edit section ***
(СтрокаЗаказа.Количество Set end)
```

Когда данное исключение будет сгенерировано, на форме будет выдано сообщение об ошибочном вводе и пока не будет введено корректное значение, пользователь не сможет покинуть поле.

Автоматическое получение списка товаров в накладной

Теперь сделаем следующую вещь: в тот момент, когда пользователь выберет нужный заказ на форме редактирования накладной, мы создадим детейлы (ЗаписьВНакладной) на основе того, что указано в заказе.

Метод, который будет вызван после срабатывания лукапа называется Edited, вот его и переопределим в независимой форме для накладной (C_НакладнаяE.cs).

Код будет следующим:

```
            // *** Start programmer edit section *** (C_НакладнаяE
CustomMembers)
        public override void Edited(ICSSoft.STORMNET.DataObject
dataobject, string contpath, string propertyname)
        {
            if (propertyname == "Заказ" &&
((Накладная)dataobject).Заказ != null)
            {
                Накладная накладная = (Накладная)dataobject;
                ICSSoft.STORMNET.Business.IDataService ds =
ICSSoft.STORMNET.Business.DataServiceProvider.DataService;
                ds.LoadObject("C_ЗаказE",накладная.Заказ);

                накладная.ЗаписьВНакладной.Clear();

                for(int i = 0; i <
накладная.Заказ.СтрокаЗаказа.Count; i++)
                {
                    ЗаписьВНакладной записьВНакладной = new
ЗаписьВНакладной();
                    записьВНакладной.Товар =
накладная.Заказ.СтрокаЗаказа[i].Товар;
                    записьВНакладной.Количество =
накладная.Заказ.СтрокаЗаказа[i].Количество;

 накладная.ЗаписьВНакладной.Add(записьВНакладной);
                }
            }
            base.Edited (dataobject, contpath, propertyname);

            ICSSoft.STORMNET.UI.BaseWinEdit frm =
(ICSSoft.STORMNET.UI.BaseWinEdit)Editor;

            frm.EditManager.Change("ЗаписьВНакладной");
        }
            // *** End programmer edit section *** (C_НакладнаяE
CustomMembers)
```

В данной функции сначала проверяем, тот ли лукап сработал, а потом с помощью сервиса данных подгружаем детейлы для класса Заявка (в подгружаемом представлении есть

детейлы), так как по умолчанию они не загружаются. После загрузки в цикле формируем и заполняем детейлы. Затем, после выполнения базовой функции, мы должны вызвать метод Change у EditManager для свойства ЗаписьВНакладной.

Организация проверки наличия товара на складе при оформлении заказа

При переводе заказа в состояние «Оплаченный» будем проверять, может ли быть товар выписан, и если да, то вычтем необходимое количество товара.

Примечание: для простоты будем считать, что товар может находиться на разных складах, и затребованное количество ищем по всем складам, суммируя.

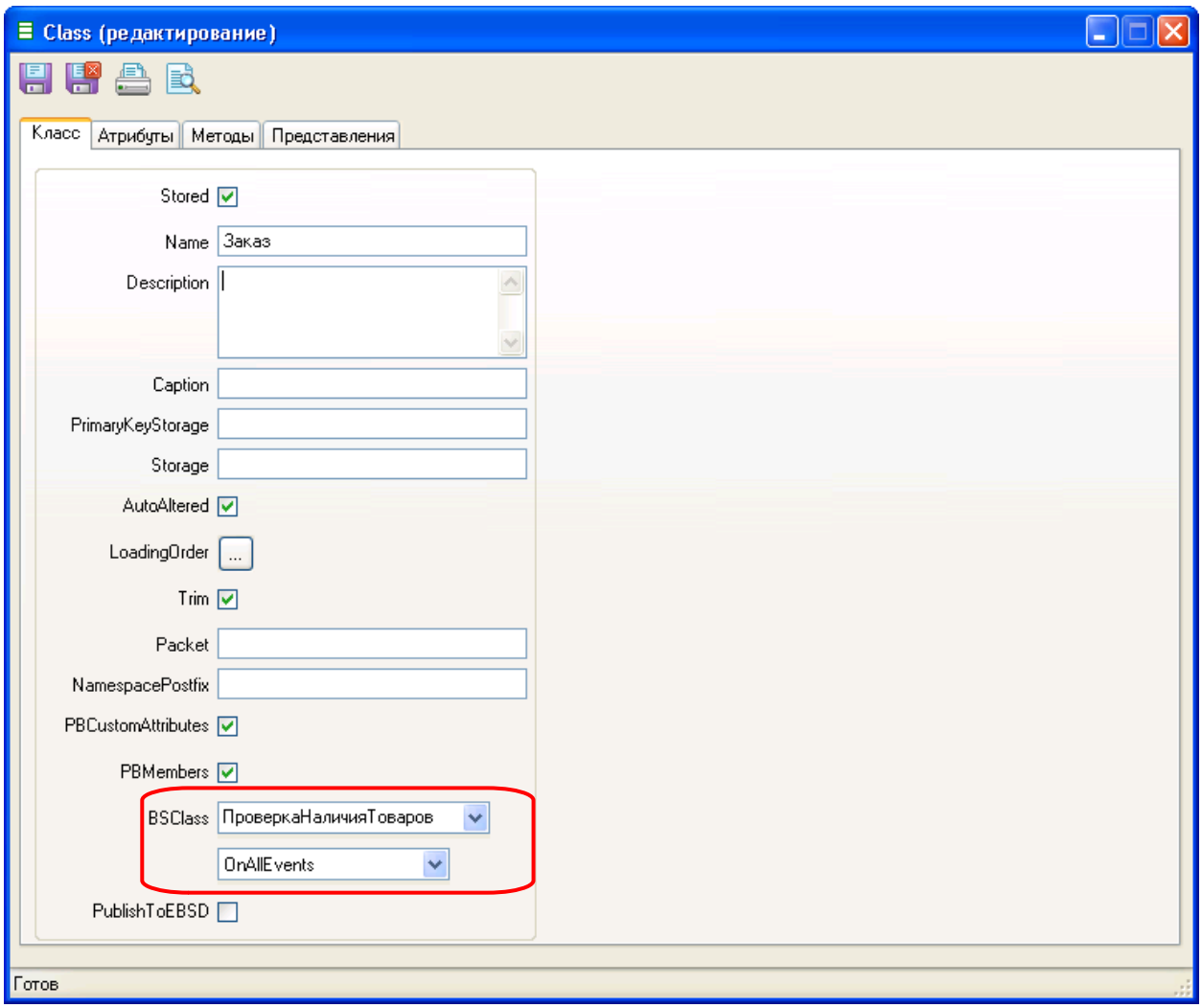
Понятие бизнес-сервера

Бизнес-сервер – это специализированный класс, который позволяет перехватить реально выполняемые сервисом данных операции над источником данных (такие как создание записи в таблице базы данных, удаление, обновление) в зависимости от состояния объекта данных. Для реализации такого класса есть стереотип «businessserver».

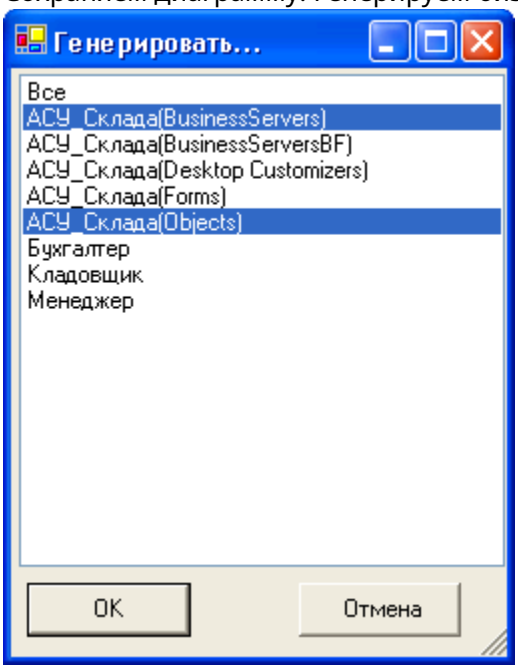
Для примера реализуем проверку наличия товара на складе в момент сохранения заказа со статусом «Оплаченный».

Рисуем класс бизнес-сервера:

Сохраним диаграмму, в свойствах класса Заказ указываем соответствующее имя бизнес-сервера и выбираем срабатывание на OnAllEvents (т.е. при любых операциях сервиса данных):



Сохраняем диаграмму. Генерируем бизнес-серверы и объекты данных:



Проект с бизнес-серверами добавляем в наш Solution. Добавляем ссылку на проект с бизнес-сервером в проекты приложений.

Подключаем сборку ICSSoft.STORMNET.FunctionalLanguage.dll к только что присоединенному проекту.

Далее в коде бизнес-сервера обрабатываем всё следующим образом:

```
            public virtual ICSSoft.STORMNET.DataObject[]
OnUpdateЗаказ(IIS.АСУ_Склада.Заказ UpdatedObject)
            {
                  // *** Start programmer edit section *** (OnUpdateЗаказ)
                  //определим массив, который будем возвращать для
обновления
                  DataObject[] ret = new DataObject[0];
                  //проверим на то, что пришедший объект - это именно то,
что нам нужно (создан или изменён и статус установлен в Оплачено)
                  if ((UpdatedObject.GetStatus() ==
ICSSoft.STORMNET.ObjectStatus.Created || UpdatedObject.GetStatus() ==
ICSSoft.STORMNET.ObjectStatus.Altered) &&
Array.IndexOf(UpdatedObject.GetAlteredPropertyNames(),"Статус") >= 0 &&
UpdatedObject.Статус == СостояниеЗаказа.Оплаченный)
                  {
                        //построим ограничение и вычитаем все объекты
ТоварНаСкладе, которые нам подходят
                        Заказ заказ = UpdatedObject;

                        SQLWhereLanguageDef langdef =
SQLWhereLanguageDef.LanguageDef;
                        ICSSoft.STORMNET.FunctionalLanguage.Function lf =
null;

                        for (int i = 0; i < заказ.СтрокаЗаказа.Count; i++)
                        {
                              if (lf != null)
                              {
                                    if(заказ.СтрокаЗаказа[i].Товар != null)
                                          lf =
langdef.GetFunction(langdef.funcOR, lf,langdef.GetFunction(langdef.funcEQ,
                                                new
VariableDef(langdef.GuidType, "Товар"),
заказ.СтрокаЗаказа[i].Товар.__PrimaryKey));
                              }
                              else
                              {
                                    if(заказ.СтрокаЗаказа[i].Товар != null)
                                          lf =
langdef.GetFunction(langdef.funcEQ,
                                                new
VariableDef(langdef.GuidType, "Товар"),
заказ.СтрокаЗаказа[i].Товар.__PrimaryKey);
                              }
                        }

                        ICSSoft.STORMNET.Business.LoadingCustomizationStruct
lcs =
ICSSoft.STORMNET.Business.LoadingCustomizationStruct.GetSimpleStruct(typeof(T
оварНаСкладе),"C_ТоварНаСкладеE");
                        lcs.LimitFunction = lf;

                        ICSSoft.STORMNET.DataObject[] objs =
ICSSoft.STORMNET.Business.DataServiceProvider.DataService.LoadObjects(lcs);
                        //разместим вычитанные объекты в отсортированном
списке для удобного доступа в дальнейшем
                        System.Collections.SortedList sl = new
System.Collections.SortedList();
                        for (int i = 0 ; i < objs.Length; i++)
                        {
                              if
(sl.ContainsKey(((ТоварНаСкладе)objs[i]).Товар.__PrimaryKey))
                              {
```

```
 ((System.Collections.ArrayList)sl[objs[i].__PrimaryKey]).Add(objs[i]);
                                }
                                else
                                {
                                      System.Collections.ArrayList
списокТоваров = new System.Collections.ArrayList();
                                      списокТоваров.Add(objs[i]);

 sl.Add(((ТоварНаСкладе)objs[i]).Товар.__PrimaryKey, списокТоваров);
                                }
                          }

                          //определим строчку для сообщения об ошибке
                          string errStr = string.Empty;
                          ArrayList retObjs = new ArrayList();

                          //проверим наличие товара на складах, если не
хватает, то выдадим сообщение об ошибке, если хватает, то вычитаем количество
                          for (int i = 0; i < заказ.СтрокаЗаказа.Count; i++)
                          {

 if(sl.ContainsKey(заказ.СтрокаЗаказа[i].Товар.__PrimaryKey))
                                {
                                      ArrayList              arl              =
((System.Collections.ArrayList)sl[заказ.СтрокаЗаказа[i].Товар.__PrimaryKey]);
                                      int количествоНаСкладах = 0;
                                      for(int j = 0; j < arl.Count; j++)
                                      {
                                            количествоНаСкладах              +=
((ТоварНаСкладе)arl[j]).Количество;
                                      }
                                      if         (количествоНаСкладах         <
заказ.СтрокаЗаказа[i].Количество)
                                      {
                                            errStr += "Не хватает товара \"" +
заказ.СтрокаЗаказа[i].Товар.Название + "\" в наличии: " + количествоНаСкладах
+ ", требуется " + заказ.СтрокаЗаказа[i].Количество + Environment.NewLine;
                                      }
                                      else
                                      {
                                            int             колич             =
заказ.СтрокаЗаказа[i].Количество;
                                            for(int j = 0; j < arl.Count; j++)
                                            {
                                                  if     (колич     >     0     &&
((ТоварНаСкладе)arl[j]).Количество > колич)
                                                  {

 ((ТоварНаСкладе)arl[j]).Количество -= колич;
                                                        колич = 0;
                                                        retObjs.Add(arl[j]);
                                                  }
                                                  else
                                                  {
                                                        if (колич > 0)
                                                        {
                                                              колич          -=
((ТоварНаСкладе)arl[j]).Количество;

 ((ТоварНаСкладе)arl[j]).Количество = 0;

 retObjs.Add(arl[j]);
```

```
                                                }
                                            }
                                        }
                                    }
                                }
                                else
                                {
                                    errStr += "Товар \"" + заказ.СтрокаЗаказа[i].Товар.Название + "\" в наличии отсутствует." + Environment.NewLine;
                                }
                            }

                            //в случае, если чего-то не хватило, сообщаем об этом пользователю
                            if (errStr != string.Empty)
                            {
                                throw new Exception(errStr);
                            }

                            //если всё нормально, то возвращаем массив объектов, которые надо обновить
                            ret = new DataObject[retObjs.Count];
                            retObjs.CopyTo(ret,0);

                        }
                        return ret;
                        // *** End programmer edit section *** (OnUpdateЗаказ)
                    }
```

Блокирование заказа, после перехода в состояние Оплачен

Чтобы избежать такой ситуации, когда количество товара на складе вычитается несколько раз, сделаем блокирование статуса заказа, если заказ уже оплачен. Перепишем метод Edit в зависимой форме:

```
                // *** Start programmer edit section *** (WinformC_ЗаказE CustomMembers)
                public override void Edit(ICSSoft.STORMNET.DataObject dataobject, string contpath, string propertyname, object tag)
                {
                    base.Edit (dataobject, contpath, propertyname, tag);

                    if (dataobject!= null)
                    {
                        EditManager.SetReadonlyFlagProperties(true, new string[]{"Цена"});
                        СтрокаЗаказа.EditManager.AfterChangeProperty -=new ICSSoft.STORMNET.Windows.Forms.Binders.AfterChangePropertyDelegate(EditManager_AfterChangeProperty);
                        СтрокаЗаказа.EditManager.AfterChangeProperty +=new ICSSoft.STORMNET.Windows.Forms.Binders.AfterChangePropertyDelegate(EditManager_AfterChangeProperty);
                        if (((Заказ)dataobject).Статус == СостояниеЗаказа.Оплаченный)
                        {
                            EditManager.SetReadonlyFlagProperties(true, new string[]{"Статус"});
                        }
```

```
            }
        }
```

Контрольные вопросы

1. Какие проекты генерируются с помощью CASEBERRY в рамках одной разрабатываемой системы?
2. Что такое вычислимый атрибут?
3. Что такое бизнес-сервер?
4. Как получить сервис данных?

Темы для самостоятельной работы

На базе полученного опыта можно дальше дорабатывать систему. Ниже представлены основные задачи, которые могут быть решены:

1. В накладной сделать вычислимые атрибуты сумма и т.д. на примере заказа и написать соответствующие обработчики.
2. Исправить сгенерированную форму таким образом, чтобы элементы управления располагались более рационально.
3. Сделать фильтрацию по кладовщикам для лукапа на форме редактирования для объектов класса Склад.

Перспективы развития АСУ_Склад

Чтобы разрабатываемая система была пригодна для использования в «боевых условиях», нужно учесть и обработать следующие моменты:

1. Пользователь должен быть ограничен в рамках жизненного цикла для заказов и накладных. То есть не должно быть произвольных переходов.
2. Документы должны выводиться на печать.
3. Срок оплаты заказа должен контролироваться.
4. Номера документов должны использоваться и как-то быть упорядочены