Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.04 Программная инженерия

**О Т Ч Е Т**

**по учебной практике**

Выполнил студент гр. РИС-19-1б

\_\_\_\_\_**\_\_\_\_**\_Рождественский А.К. \_**\_\_\_\_\_\_**

(фамилия Имя Отчество)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(подпись)

Проверил:

*\_\_\_старший преподаватель кафедры ИТАС Кузнецов Д.Б.\_\_*

*(должность, ФИО руководителя по практической подготовке от кафедры)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(оценка) (подпись)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(дата)*

**Пермь 2021**

Содержание

[Введение 3](#_Toc76955985)

[1 Анализ предметной области 4](#_Toc76955986)

[1.1 Ознакомление с интерфейсом SAP 4](#_Toc76955987)

[1.2 Ознакомление с языком программирования ABAP 6](#_Toc76955988)

[1.3 Анализ поставленной задачи и способы ее реализации на языке ABAP 9](#_Toc76955989)

[2 Технология реализации приложения на языке ABAP 11](#_Toc76955990)

[2.1 Создание объектов базы данных 11](#_Toc76955991)

[2.2 Создание классов для работы с данными 13](#_Toc76955992)

[2.3 Создание дополнительных экранов и работы с ними 17](#_Toc76955993)

[Заключение 19](#_Toc76955994)

[Список использованных источников 20](#_Toc76955995)

[Приложение А. 21](#_Toc76955996)

# 

Введение

ABAP/4 – это язык программирования высокого уровня, созданный компанией SAP в 1980 году. Аббревиатура ABAP расшифровывается как Advanced Business Application Programming, данный язык создавался для разработки бизнес-приложений в среде разработки SAP. Данная платформа так же поддерживает разработку на языке Java. ABAP позволяет реализовать как объектно-ориентированное программирование, основанное на классах и интерфейсах, так и процедурное, основанное на функциональных модулях и подпрограммах. Так как основным назначением данного языка является разработка бизнес-приложений для коммерческих компаний, то неотъемлемой частью ABAP, как языка программирования, является полный доступ к базе данных, где данный доступ интегрирован в язык.

Цель данной учебной практики реализовать систему учета продажи автомобилей.

Для выполнения заданной цели решим поэтапно следующие задачи:

1. Провести анализ предметной области.
2. Разработка структуры программы.
3. Разработка приложения для работы с базой данных.

# 1 Анализ предметной области

Предметной областью у нас является создание системы учета продажи автомобилей в среде разработки SAP на языке программирования ABAP. Так как основной целью не является построение базы данных, значит для примера работы программы будет достаточно 2 сущности: Автомобили и покупатели.

Данные сущности будут созданы, используя встроенные в язык программирования инструменты.

* 1. Ознакомление с интерфейсом SAP

Одно из главных отличий ABAP от других языков программирования как C++, Java и Python является то, что код программы никогда не хранится во внешних файлах, он находится в базе данных SAP и при запуске программы требуется войти в сеть предприятия, работающего с данной средой. На рисунке 1 изображено окно входа в систему.

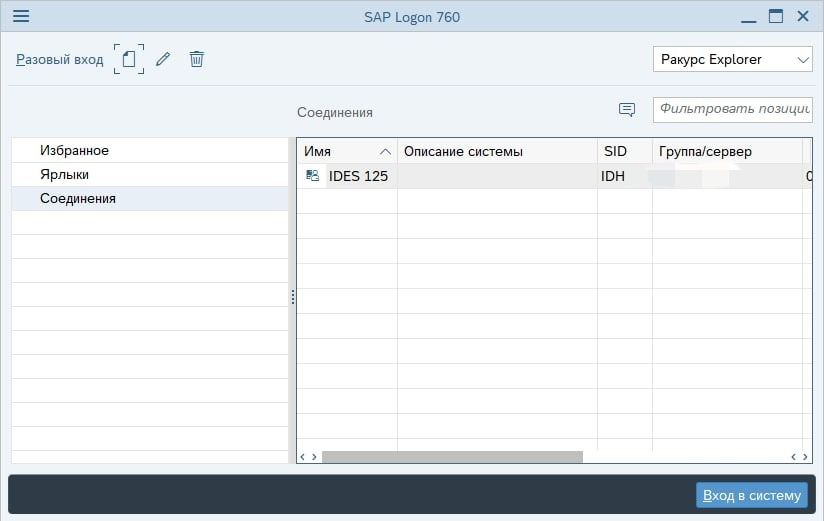


Рисунок 1 - Начальный экран

В данном случае, подключение уже настроено и готово к работе после входа в систему, используя учетную запись, выданную на предприятии.

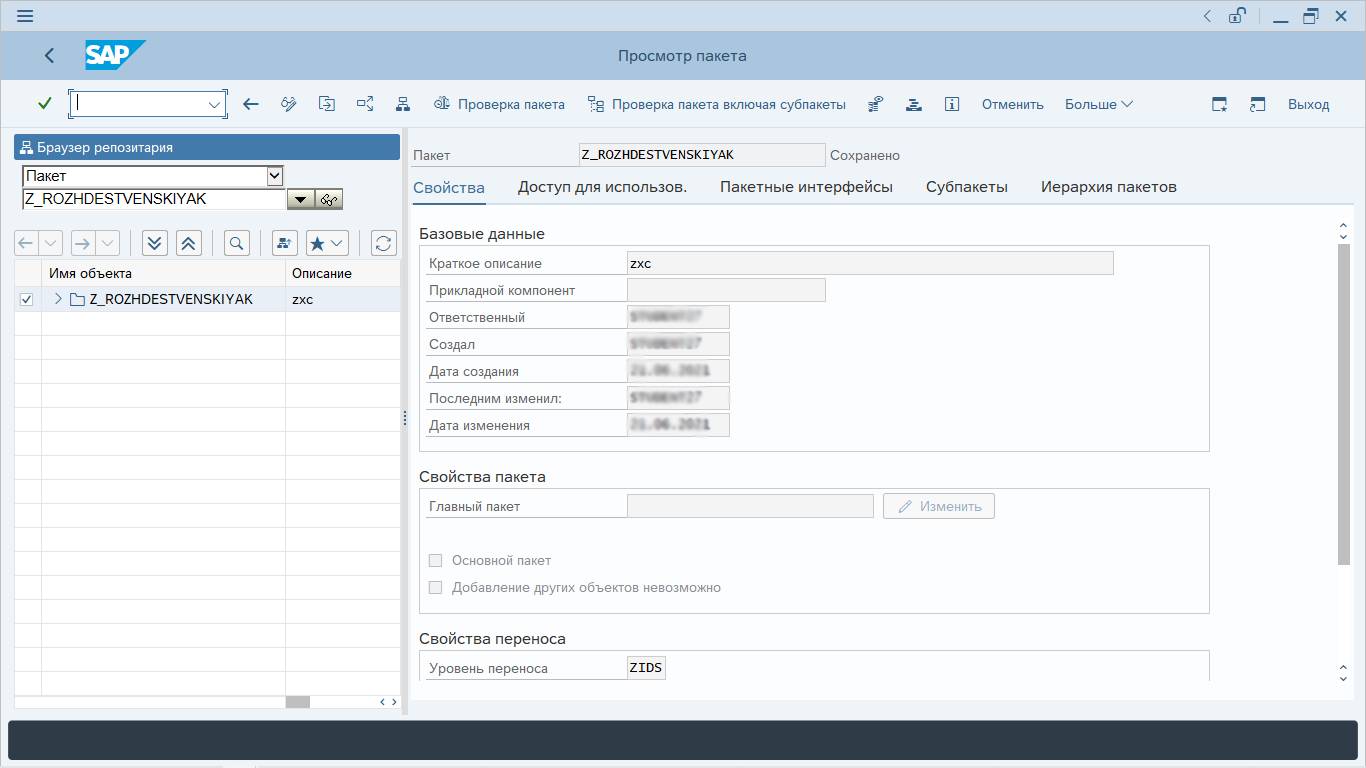
После входа в систему открывается окно с репозитарием в котором мы можем создавать пакеты и другие виды объектов, для хранения их на сервере. Окно с репозитарием изображено на рисунке 2. В данном случае создается пакет, в котором будет храниться сам код программы и глобальные классы. 

Рисунок 2 - Окно репозитария.

Структура пакета похожа на структуру хранения файлов, в данном случае у нас хранятся несколько папок: Объекты словаря, Библиотека классов, Программы, Группы функций и Includes. Структура пакета изображена на рисунке 3.

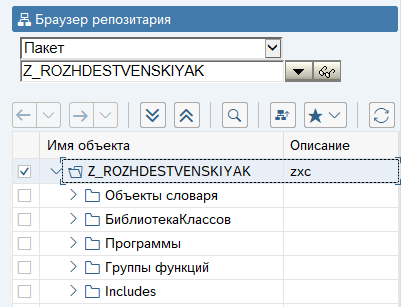


Рисунок 3 - Структура пакета.

В объектах словаря хранятся таблицы, структуры данных, типы таблиц и элементы данных, так как программа подразумевает использование базы данных, то в данной папке будут созданы таблицы и необходимое количество элементов данных, выявленных в дальнейшем по ходу анализа. Классы, программы будут созданы по такой же логике.

Как итог, прошло ознакомление с интерфейсом среды разработки и можно приступить к изучению самого языка программирования ABAP.

* 1. Ознакомление с языком программирования ABAP

ABAP – проприетарный язык программирования высокого уровня, используемый в решениях компании SAP. Как и в других языках программирования весь код тут разветвляется на исходный код, написанный программистом для реализации поставленной задачи на данном языке и внутренний код, который является частью ядра SAP, написанный для контролирования логики и реакции на события. Перейдем к типам данных, в данном языке они мало чем отличаются от их реализации в других языках программирования. В данном языке типы данных различаются на полные и неполные, различие их состоит в том, что полные типы данных имеют заранее определенную фиксированную длину. В список полных стандартных типов данных входят:

1. D – тип для даты. Формат ГГГГММДД, длиной 8.
2. T – тип для времени. Формат ЧЧММСС, длиной 6.
3. I – тип для целого числа, длиной 4.
4. F – тип для числа с плавающей точкой, длиной 8.
5. STRING – тип для символьной строки переменной длины.

Неполные стандартные типы данных:

1. С – тип для символьной строки, для которой нужно указать длину (Пример инициализации переменной длиной 20 символов: С20).
2. N – тип для числовой символьной строки, для которой нужно указать длину.
3. X – тип последовательности байтов, для которой нужно указать длину.

Стандартные типы данных можно использовать в программе для описания локальных типов данных, которые могут быть более полными или сложными, чем лежащие в их основе стандартные типы. Локальные типы данных существуют только в отдельно взятой программе и, следовательно, могут использоваться только в ней. Для описания типов используется оператор TYPES. Циклы и ветвления смысла рассматривать нет, в данном языке они такие же, как и в популярных языках программирования. В данном языке используется конструкция с условием IF, CASE, DO и WHILE. Синтаксис изображен на рисунке 4.

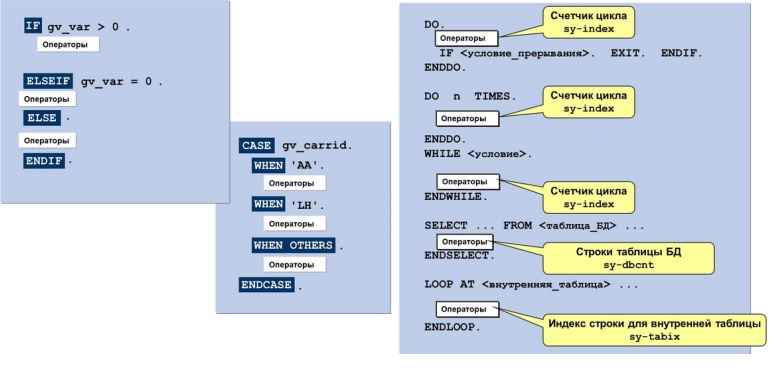


Рисунок 4 - Варианты ветвления кода.

Для модуляризации программы в данном языке используются функции, объявляются которые в основном коде программы с помощью конструкции FORM и ENDFORM и вызываются с помощью PERFORM. Примерная конструкция изображена на рисунке 5.

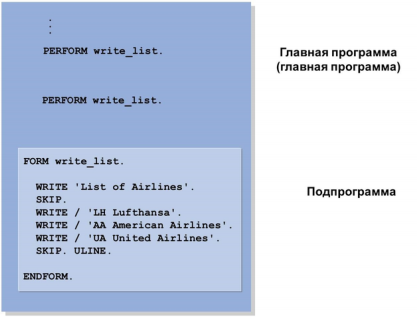


Рисунок 5 - Пример функции.

Так же для работы и масштабирования программы потребуется создать классы для работы с данными. В ABAP-редакторе, который является IDE (Интегрированной средой разработки) есть возможность определения глобальный и локальных классов. Локальные классы определяется с помощью описания класса в отдельном файле и вызываемом с помощью INCLUDE, но для реализации поставленной задачи локальные классы не используются. Вся работа с данными происходит в глобальных классах, которые создаются с помощью встроенного конструктора, изображенного на рисунке 6.

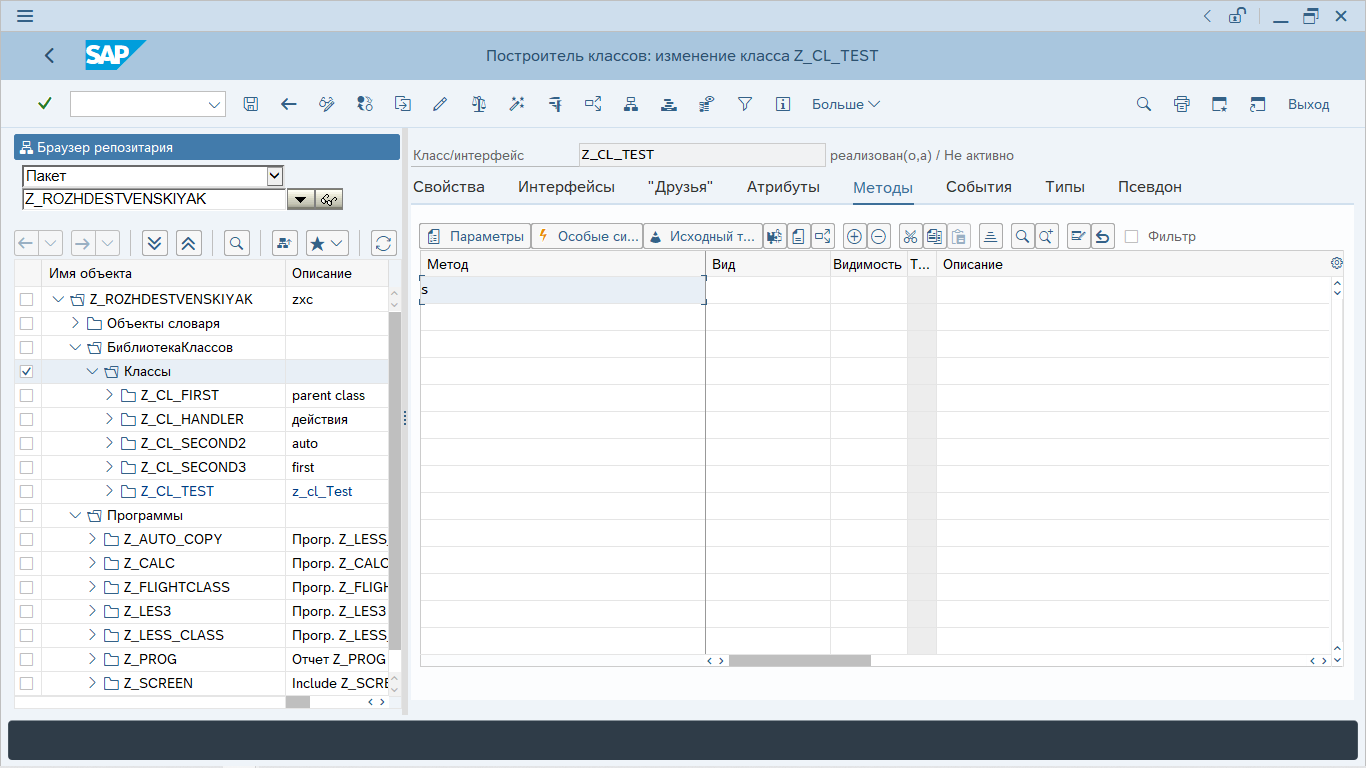


Рисунок 6 - Конструктор классов.

В данном окне можно увидеть вкладки, в котором и описывается вся логика программы, а самое главное это атрибуты и методы класса, объявлять которые мы будем в разделе технологии реализации.

Выводиться результаты работы программы будут в экраны. Экран – это локальный объект, который может содержать в себе структуру с полями ввода/вывода, кнопками и другими элементами экрана, помимо этого в экране так же содержится логика его обработки. В нашем случае экран будет содержать только контейнер для вывода ALV (ABAP LIST VIEWER). Данный элемент экрана используется для вывода таблицы на экран.

По итогу был изучен синтаксис языка ABAP и реализацию элементов для выполнения задания.

* 1. Анализ поставленной задачи и способы ее реализации на языке ABAP

Создание учета продажи автомобилей в среде разработки SAP – задача на реализацию данной учебной практикой от выбранного предприятия. Так как в первую очередь язык ABAP предназначен для работы с таблицами, то требуется создать схему отношений для работы с данными. В нашем случае достаточно будет двух таблиц – это таблица с автомобилями и покупателями. Модель данных изображена на рисунке 7.

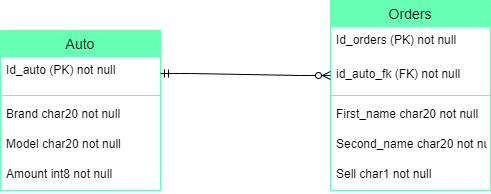


Рисунок 7 - Модель данных.

Для работы с данными было выбрано использовать классы, чтобы лучше ознакомиться с языком программирования на практике, всего будет создано 3 класса. Родительский класс, содержащий в себе методы отрисовки ALV и обращения к классу событий. От родительского класса наследуется класс автомобили, в котором будут содержаться методы обработки данных из таблицы автомобилей.

Результаты работы программы так же было выбрано выводить в отдельное окно для красивого вывода таблиц и углубления в изучении языка. Так же причиной такого выбора послужило отсутствие других способов вывода таблиц на экран.

Вывод: По результатам анализа был изучен интерфейс программы ABAP Workbench, синтаксис языка ABAP и проанализирована поставленная задача и выбран способ её решения.

# Технология реализации приложения на языке ABAP

В данном разделе будет рассмотрена реализация написанной в процессе практики программы, а именно создание объектов хранения данных, обработки и вывода итогов работы программы конечному пользователю.

## Создание объектов базы данных

Путем анализа было выявлено, что для реализации программы потребуется 2 таблицы: Автомобили и заказы. Так как таблица автомобили у нас является основной, а заказы связана с ней по внешнему ключу, то первым делом создадим таблицу автомобили. Создаётся она с помощью встроенного в IDE конструктора. После создания требуется объявить поля данной таблицы. Поля представлены в модели данных. Так как атрибуты таблицы содержат нестандартные типы данных, а именно символьный тип с длиной 20 символов, то потребуется так же создать собственные типы данных для каждого атрибута отношения с помощью конструктора. Пример одного из используемых типов данных представлен на рисунке 8.

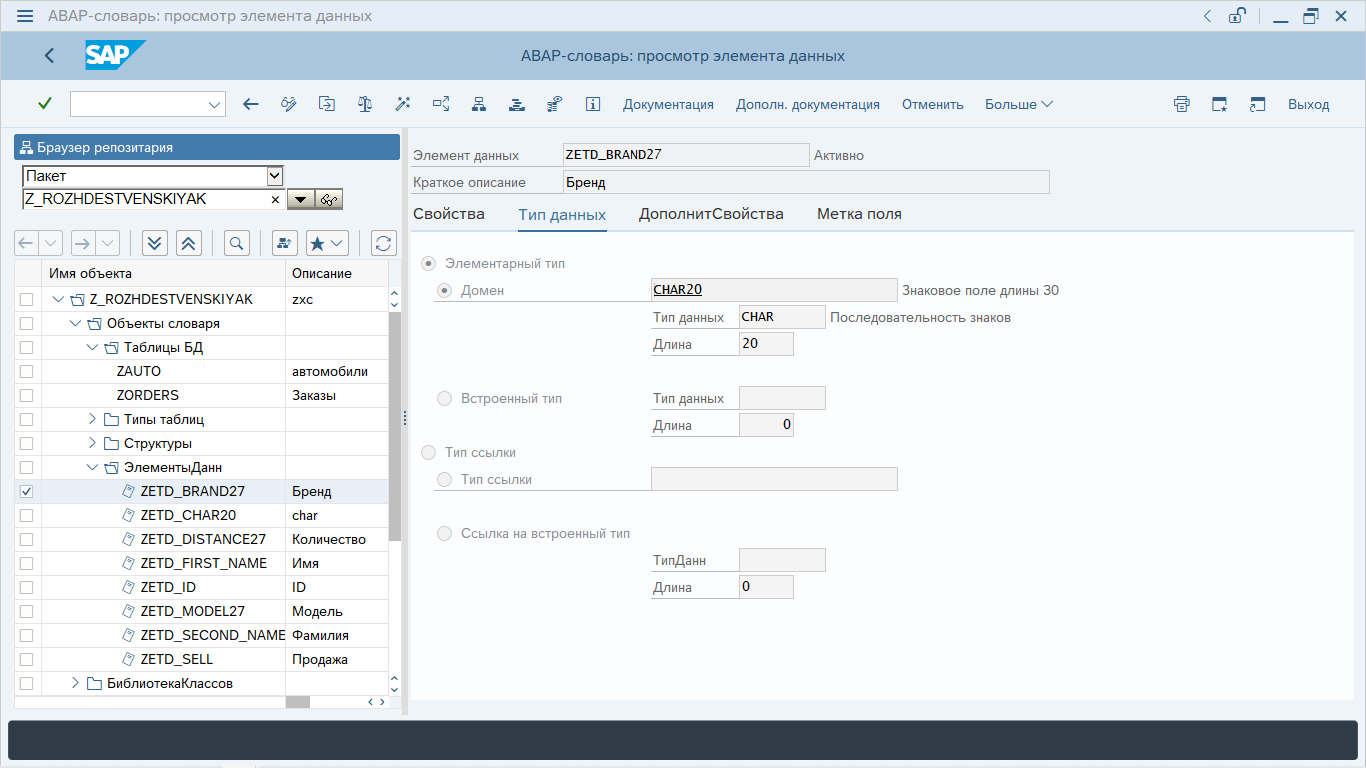


Рисунок 8 - Собственный тип данных

После создания всех типов данных можно приступить к созданию таблиц. Готовая таблица изображена на рисунке 9.

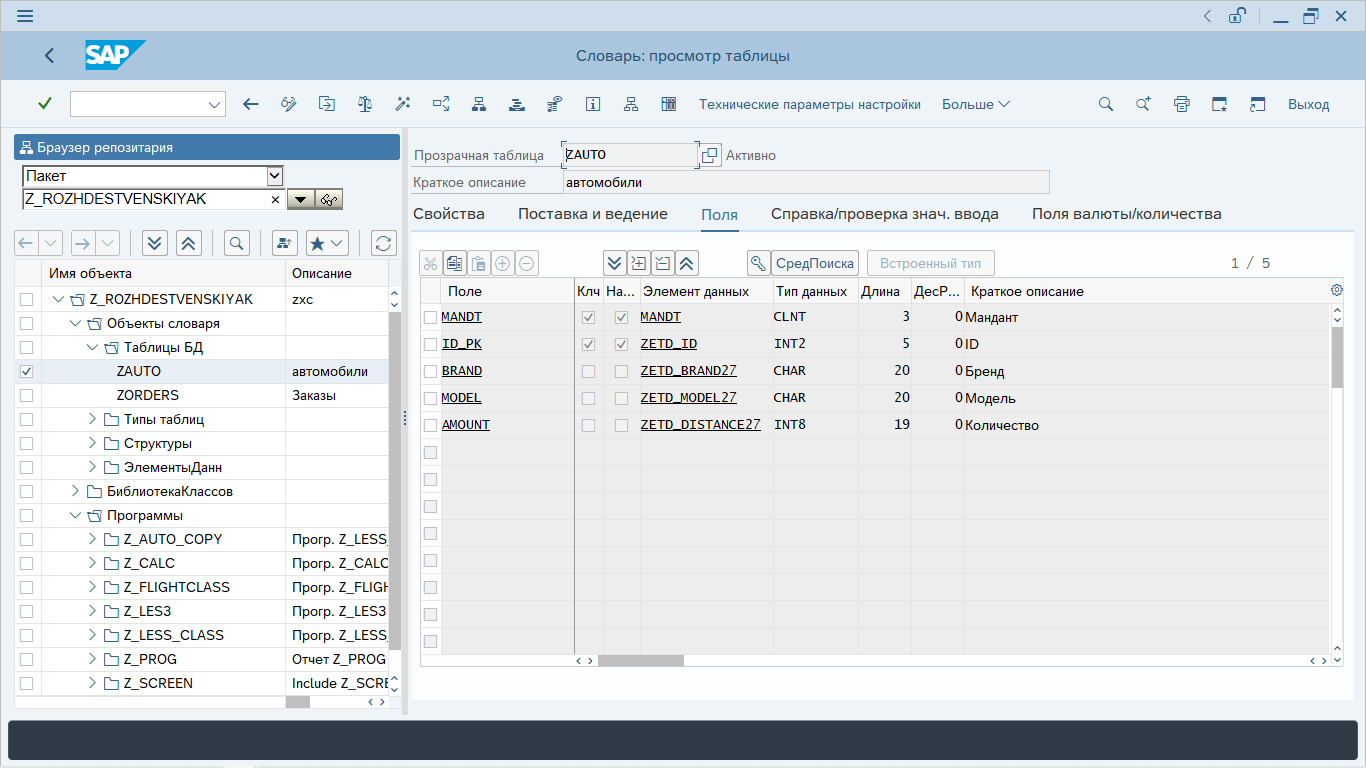


Рисунок 9 - Таблица автомобили.

По такой же логике создаем таблицу заказы. Создается объект и объявляются поля по построенной модели данных. Готовая таблица изображена на рисунке 10.

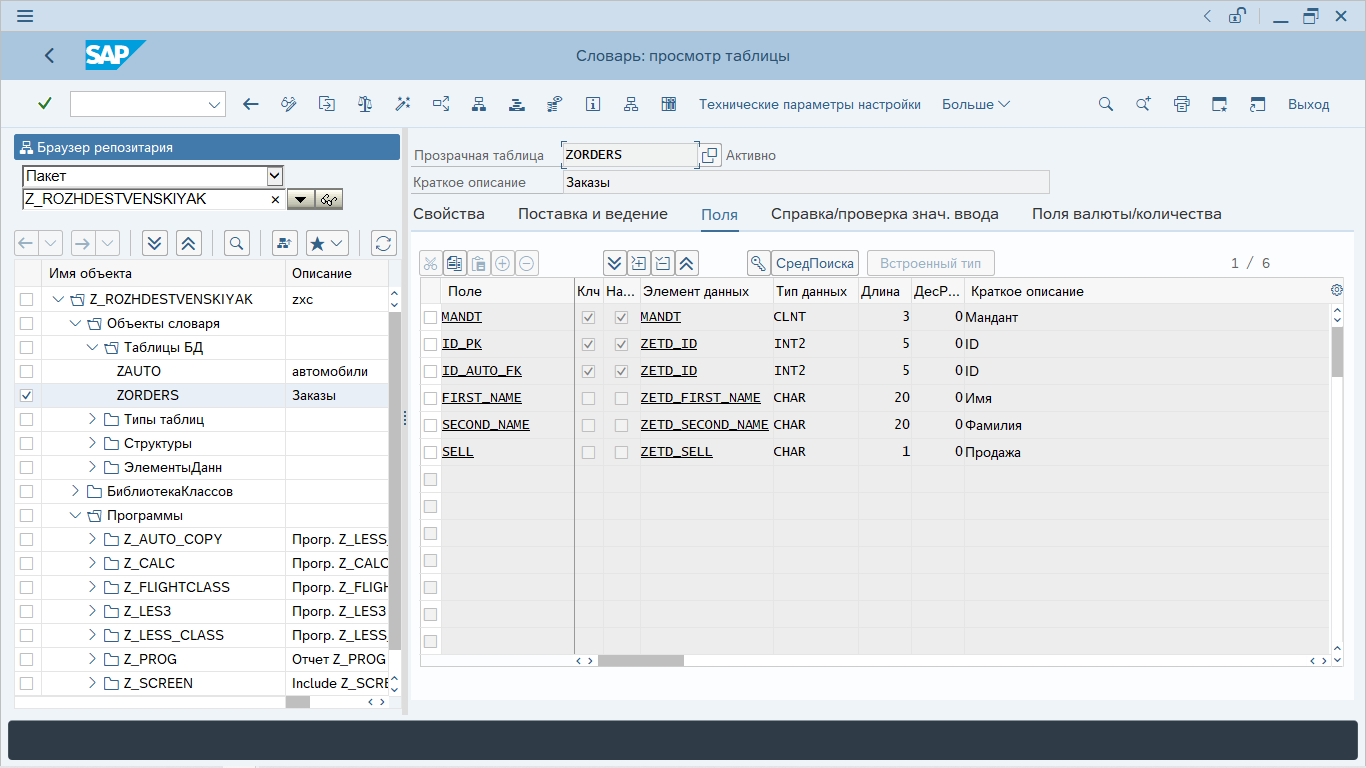


Рисунок 10 - Таблица заказы.

## Создание классов для работы с данными

Работа программы будет реализована с помощью глобальных классов. Такого типа классы так же создаются с помощью конструктора, после создания класса требуется заполнить его атрибуты и используемые методы. В программе у нас используется наследование классов, поэтому первым делом создаем родительский класс. В котором будет описана логика создания ALV (вывод таблиц) и FIELD CATALOG (задания параметров атрибутам отношений). Атрибуты родительского класса изображены на рисунке 11, методы на рисунке 12.

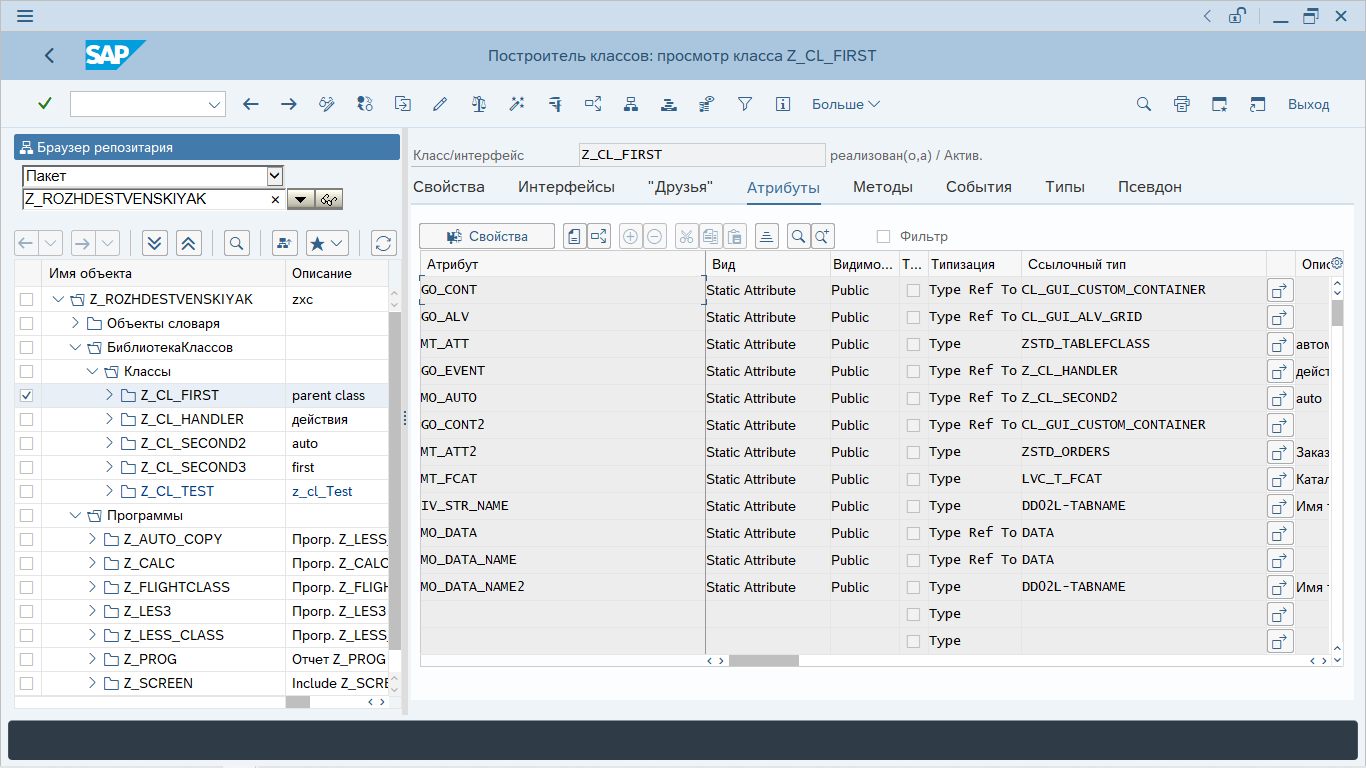


Рисунок 11 – Атрибуты родительского класса.

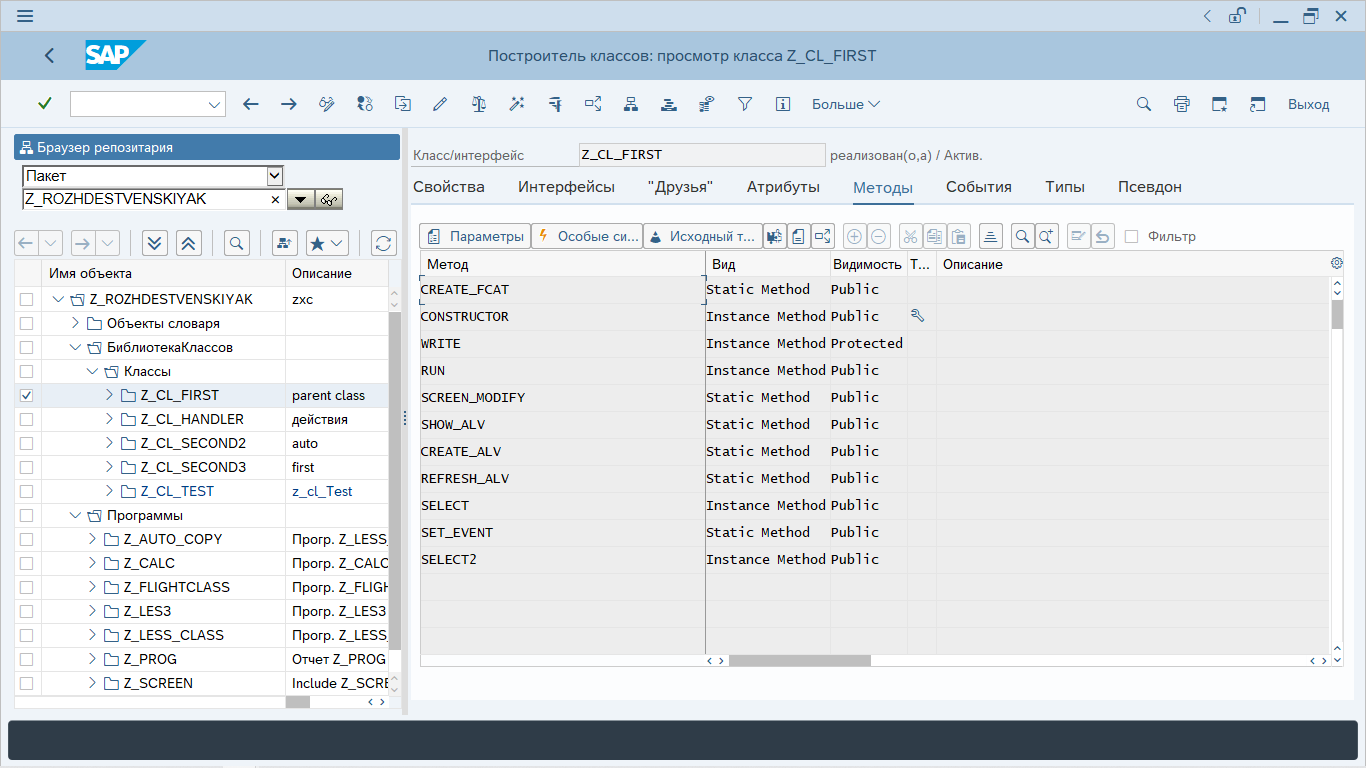


Рисунок 12 – Методы родительского класса.

Методы CREATE\_ALV, SHOW\_ALV, REFRESH\_ALV и SCREEN\_MODIFY используются для отрисовки самой таблицы в ALV на экране. CREATE\_FCAT создает объект, который задает полям условия, в нашем случае он добавляет чекбокс в атрибут таблицы. Листинг всех методов описан в Приложении А.

После создания родительского класса можно приступить к созданию наследуемого от него класса. Наследуемый класс будет называться автомобили и предназначаться для работы с таблицами. Атрибуты и методы представлены на рисунках 13 и 14 соответственно.

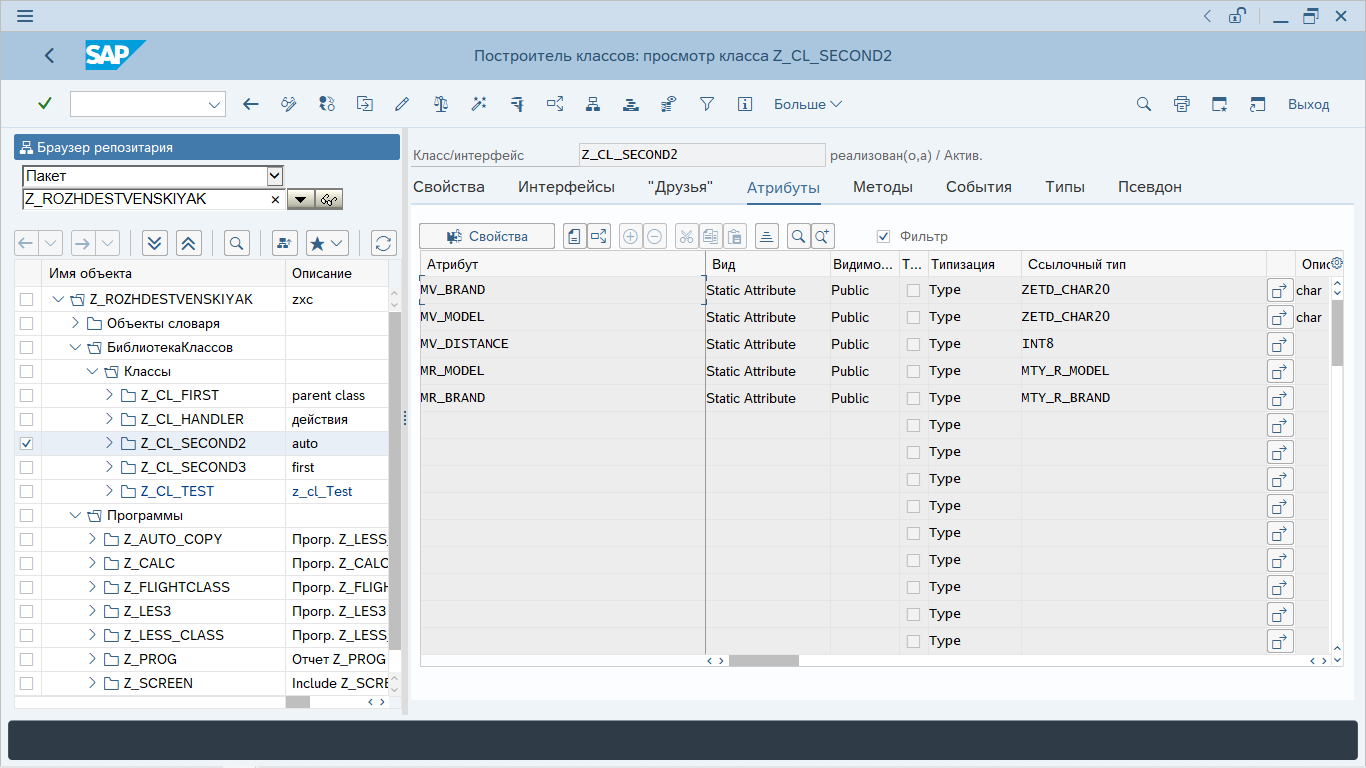


Рисунок 13 – Атрибуты класса автомобили.

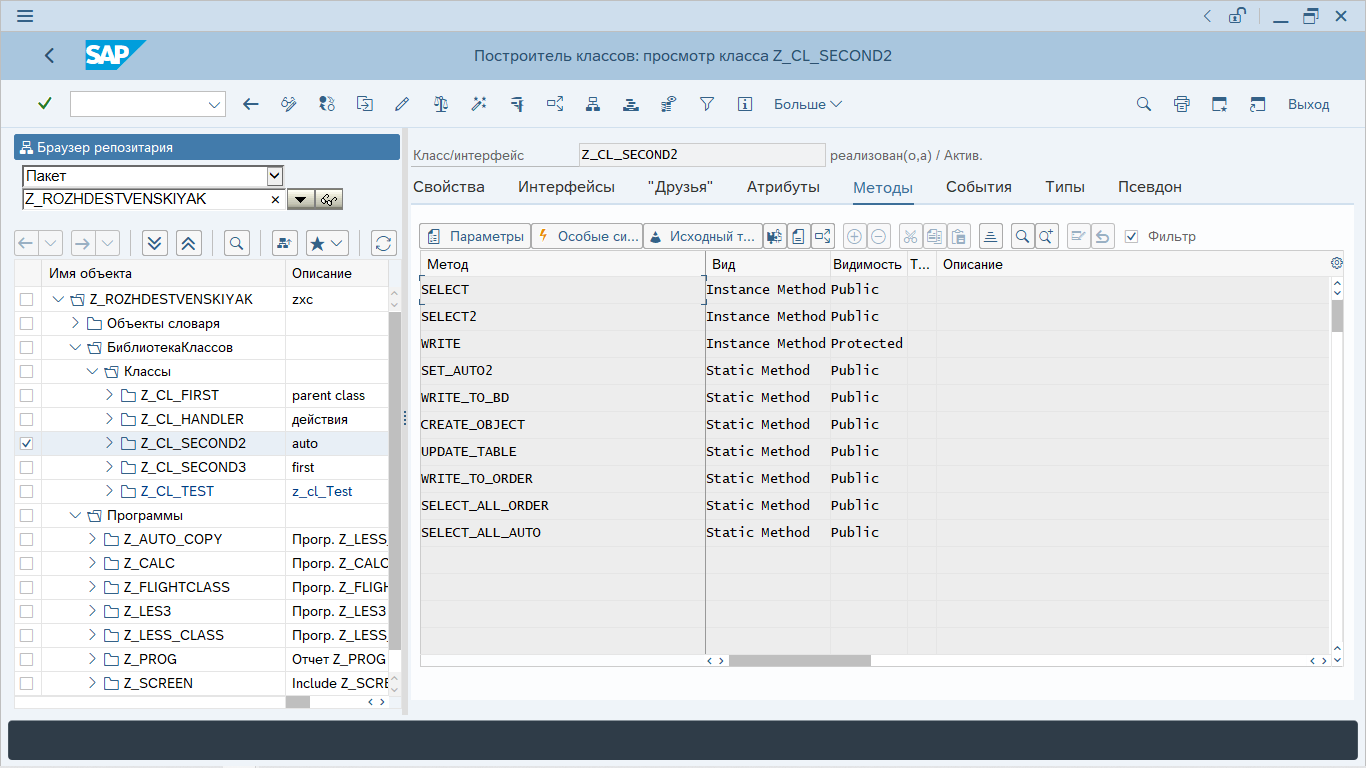


Рисунок 14 – Методы класса автомобили.

В методе SELECT и SELECT2 выполняется выборка из таблиц автомобили и заказы соответственно. SET\_AUTO2 считывает данные с полей, находящихся на селекционном экране, и, если они заполнены, то производит выборку из таблицы по заполненным значениям, иначе выводит таблицу полностью. WRITE\_TO\_BD и WRITE\_TO\_ORDER записывает данные в таблицу при нажатии кнопки добавления записи в панели инструментов. SELECT\_ALL\_ORDER и SELECT\_ALL\_AUTO используется для переключения между таблицами, если этого потребует пользователь путем нажатия кнопки. Листинг методов так же описан в Приложении А. Основные классы работы с данными созданы. Приступим к созданию класса обработчика событий. Его задача состоит в том, чтобы при двойном клике на строку с автомобилем выводить таблицу покупок этого автомобиля. Так же требуется добавить кнопки добавления, удаления и смены таблицы на панель инструментов. Для выполнения события двойного клика создается метод ON\_DOUBLE\_CLICK. Кнопки добавляются в методе ON\_TOOLBAR, а обработка нажатий на эти кнопки в USER\_COMMAND. Все методы представлены на рисунке 15.

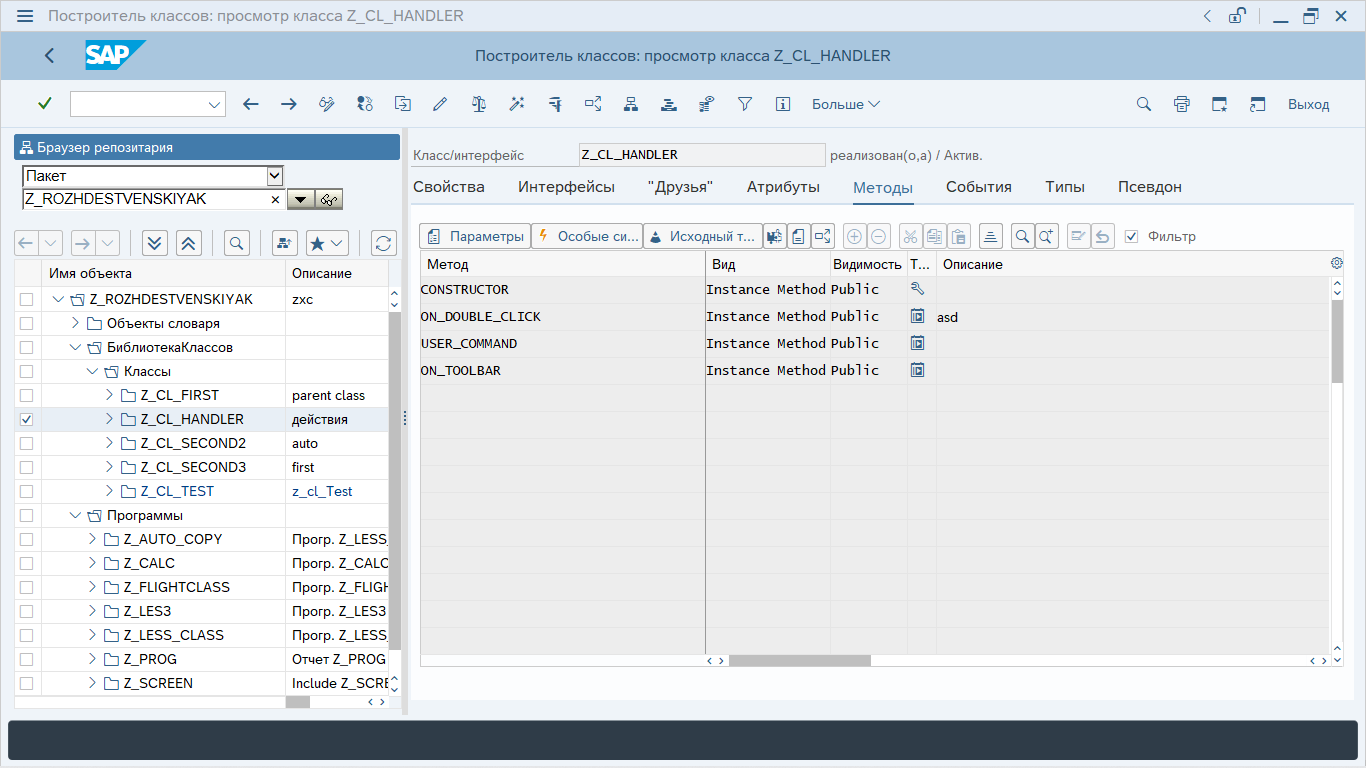


Рисунок 15 – Методы класса обработчика событий.

Листинг этого класса так же будет представлен в Приложении А.

Как итог классы для работы с данными созданы и можно приступить к созданию экранов вывода таблиц.

## Создание дополнительных экранов и работы с ними

С помощью конструктора создаем экран для вывода ALV с таблицами. С помощью редактора добавляем на весь экран контейнер, который и будет принимать в себя таблицу. В логике выполнения экрана описываем модуль для работы с кнопками и выводом ALV. Для ввода новых значений в таблицы потребуется реализовать дополнительные экраны с полями таблицы, изображенные на рисунке 16.

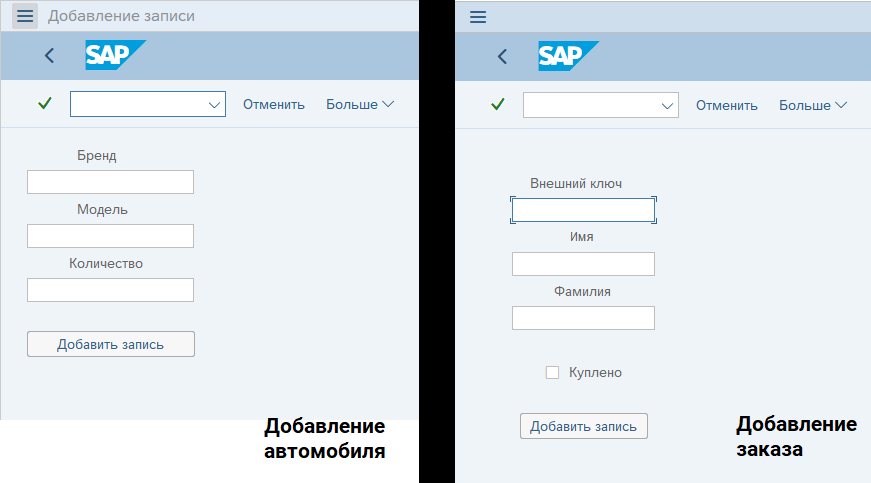


Рисунок 16 – Экраны добавления записи в таблицы.

Как итог все необходимые объекты созданы и можно приступить к их связыванию в основном файле программы.

Основной блок программы делится на модули: INITIALIZATION, AT SELECTION-SCREEN и START-OF-SELECTION. Перед написанием кода в этих модулях необходимо объявить переменные, с которыми и будет происходить работа в данных блоках. В блоке INITIALIZATION мы задаем название кнопки в панели инструментов, в нашем случае это будет кнопка с выводом автомобилей. После нажатия на кнопку происходит обработка действий, заданных в блоке AT SELECTION-SCREEN, тут мы создаем объект класса автомобили и производим выборку из таблицы с выводом на экран с объявленной ALV. В последнем блоке выполняется вся оставшаяся логика, а это: работа экранов и действия при нажатии кнопок.

Как итог получаем следующую программу, изображенную на рисунках ниже.

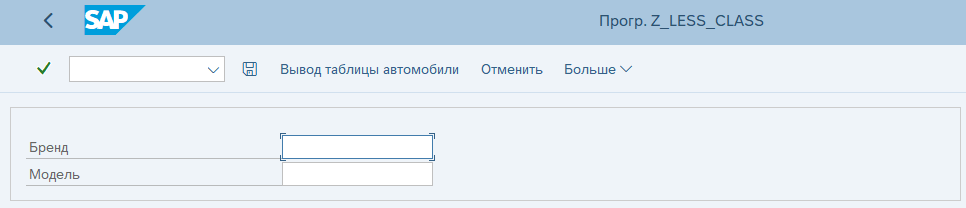
с

Рисунок 17 – Стартовое окно программы.

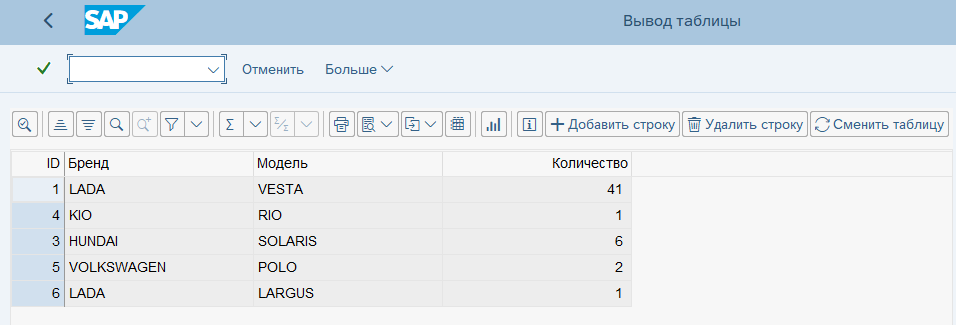


Рисунок 18 – Окно вывода таблицы автомобили.

Вывод: была реализована программа учета продажи автомобилей на языке ABAP в среде разработки SAP.

# Заключение

В ходе выполнения учебной практики был изучен интерфейс программы ABAP Workbench, синтаксис языка программирования ABAP и проведен анализ поставленной задачи.

Так же была создана база данных для демонстрации работы программы, экраны для вывода таблиц базы данных, классы для работы с данными таблиц.

Поставленную задачу – Реализации учета продажи автомобилей можно считать выполненной.

Данная программа может использоваться в дилерских центрах для организации внутренней системы контроля продаж автомобилей, а так как данная программа выполнена с использованием принципов ООП, то в дальнейшем она может быть расширена, если того потребует пользователь.

# Список использованных источников

1. *Хомоненко А. Д. Базы данных: Учебник для высших учебных заведений/ В.М. Цыганков, М. Г. Мальцев, под ред. проф. А. Д. Хомоненко— СПб.: КОРОНА принт, 2002 — 672 с.*
2. *Кречмер Р. Разработка приложений SAP R/3 на языке ABAP/4/ В.Вейс, под ред. Кречмер Р. – Берлин: Лори, 1998 – 348 с.*
3. *Точенюк О. Рекомендации по оптимизации программ на языке ABAP. – Москва: Эксперт РП, 2013 – 152с.*
4. *Мартин Р. Чистый код. Создание, анализ и рефакторинг. – Паль-Альто: Питер, 2010 – 464 с.*

# Приложение А.

Основной код

*\*&---------------------------------------------------------------------\**  
*\*& Report Z\_LESS\_CLASS*  
*\*&---------------------------------------------------------------------\**  
*\*&*  
*\*&---------------------------------------------------------------------\**  
REPORT z\_auto\_copy.  
TABLES: sscrfields.  
  
DATA: gs\_auto TYPE zauto.  
  
SELECTION-SCREEN BEGIN OF BLOCK part1 WITH FRAME.  
SELECT-OPTIONS: so\_brand FOR gs\_auto-brand NO-EXTENSION NO INTERVALS ,  
                so\_model FOR gs\_auto-model NO-EXTENSION NO INTERVALS .  
SELECTION-SCREEN END OF BLOCK part1.  
  
  
SELECTION-SCREEN FUNCTION KEY 1.  
SELECTION-SCREEN FUNCTION KEY 2.  
  
DATA: ls\_table   TYPE zauto,  
      go\_alv     TYPE REF TO cl\_gui\_alv\_grid,  
      go\_cont    TYPE REF TO cl\_gui\_custom\_container,  
      gs\_zauto   TYPE zauto,  
      gt\_auto    TYPE TABLE OF zauto,  
      go\_event   TYPE REF TO z\_cl\_handler,  
      go\_check   TYPE i,  
      gs\_zorders TYPE zorders  
      .  
  
DATA: go\_auto      TYPE REF TO z\_cl\_second2,  
      is\_transport TYPE  zauto.  
  
  
INITIALIZATION.  
  
  sscrfields-functxt\_01 = 'Вывод таблицы автомобили'.  
  
  
  DATA itab TYPE TABLE OF sy-ucomm.  
  
  APPEND: 'ONLI' TO itab.  
  CALL FUNCTION 'RS\_SET\_SELSCREEN\_STATUS'  
    EXPORTING  
      p\_status  = sy-pfkey  
    TABLES  
      p\_exclude = itab.  
  
  
  
AT SELECTION-SCREEN.  
  
  IF sy-ucomm <> 'CMD' .  
    z\_cl\_second2=>create\_object( )->set\_auto2(  
    EXPORTING  
      rv\_brand = so\_brand[]  
      rv\_model = so\_model[]  
      rt\_auto = gt\_auto  
       ).  
  
    CASE sscrfields-ucomm.  
      WHEN 'FC01'.  
        z\_cl\_second2=>create\_object( )->select( ).  
        go\_check = 1.  
        CALL SCREEN 100.  
  
    ENDCASE.  
  ENDIF.  
  
  
START-OF-SELECTION.  
  
*\*&---------------------------------------------------------------------\**  
*\*&      Module  STATUS\_0100  OUTPUT*  
*\*&---------------------------------------------------------------------\**  
*\*       text*  
*\*----------------------------------------------------------------------\**  
MODULE status\_0100 OUTPUT.  
  SET PF-STATUS 'STATUS100'.  
  SET TITLEBAR 'TITLE100'.  
ENDMODULE.  
*\*&---------------------------------------------------------------------\**  
*\*&      Module  USER\_COMMAND\_0100  INPUT*  
*\*&---------------------------------------------------------------------\**  
*\*       text*  
*\*----------------------------------------------------------------------\**  
MODULE user\_command\_0100 INPUT.  
  
  DATA: ok\_code TYPE sy-ucomm.  
  CASE ok\_code.  
    WHEN 'BACK'.  
      LEAVE TO SCREEN 0.  
    WHEN 'EXIT'.  
      LEAVE PROGRAM.  
    WHEN OTHERS.  
  ENDCASE.  
  
  
ENDMODULE.  
*\*&---------------------------------------------------------------------\**  
*\*&      Module  STATUS\_0200  OUTPUT*  
*\*&---------------------------------------------------------------------\**  
*\*       text*  
*\*----------------------------------------------------------------------\**  
MODULE status\_0200 OUTPUT.  
  SET PF-STATUS 'STATUS200'.  
  SET TITLEBAR 'TITLE200'.  
ENDMODULE.  
*\*&---------------------------------------------------------------------\**  
*\*&      Module  USER\_COMMAND\_0200  INPUT*  
*\*&---------------------------------------------------------------------\**  
*\*       text*  
*\*----------------------------------------------------------------------\**  
MODULE user\_command\_0200 INPUT.  
  CASE ok\_code.  
    WHEN 'BACK'.  
      LEAVE TO SCREEN 0.  
    WHEN 'EXIT'.  
      LEAVE PROGRAM.  
    WHEN 'FC03'.  
      DATA lv\_id TYPE i.  
      SELECT MAX( id\_pk )  
        FROM zauto  
        INTO lv\_id.  
      ls\_table-id\_pk = lv\_id + 1 .  
      ls\_table-brand = gs\_zauto-brand.  
      ls\_table-model = gs\_zauto-model.  
      ls\_table-amount = gs\_zauto-amount.  
      z\_cl\_second2=>create\_object( )->write\_to\_bd( ls\_table ).  
  
    WHEN OTHERS.  
  ENDCASE.  
  
ENDMODULE.  
*\*&---------------------------------------------------------------------\**  
*\*&      Module  SHOW\_ALV  OUTPUT*  
*\*&---------------------------------------------------------------------\**  
*\*       text*  
*\*----------------------------------------------------------------------\**  
MODULE show\_alv OUTPUT.  
  
  z\_cl\_second2=>create\_object( )->show\_alv( ).  
  
ENDMODULE.  
*\*&---------------------------------------------------------------------\**  
*\*&      Module  ADD\_AUTO  OUTPUT*  
*\*&---------------------------------------------------------------------\**  
*\*       text*  
*\*----------------------------------------------------------------------\**  
  
FORM use\_screen.  
  
  CALL SCREEN 200.  
  
ENDFORM.  
*\*&---------------------------------------------------------------------\**  
*\*&      Module  STATUS\_0300  OUTPUT*  
*\*&---------------------------------------------------------------------\**  
*\*       text*  
*\*----------------------------------------------------------------------\**  
MODULE status\_0300 OUTPUT.  
  SET PF-STATUS 'STATUS300'.  
  SET TITLEBAR 'TITLE300'.  
ENDMODULE.  
*\*&---------------------------------------------------------------------\**  
*\*&      Module  USER\_COMMAND\_0300\_EC  INPUT*  
*\*&---------------------------------------------------------------------\**  
*\*       text*  
*\*----------------------------------------------------------------------\**  
MODULE user\_command\_0300\_ec INPUT.  
  
  CASE ok\_code.  
    WHEN 'BACK'.  
      LEAVE TO SCREEN 0.  
    WHEN 'EXIT'.  
      LEAVE PROGRAM.  
  ENDCASE.  
  
ENDMODULE.  
*\*&---------------------------------------------------------------------\**  
*\*&      Module  SHOW\_ALV2  OUTPUT*  
*\*&---------------------------------------------------------------------\**  
*\*       text*  
*\*----------------------------------------------------------------------\**  
MODULE show\_alv2 OUTPUT.  
  
  z\_cl\_second2=>create\_object( )->show\_alv( ).  
  
ENDMODULE.  
*\*&---------------------------------------------------------------------\**  
*\*&      Module  STATUS\_0210  OUTPUT*  
*\*&---------------------------------------------------------------------\**  
*\*       text*  
*\*----------------------------------------------------------------------\**  
MODULE status\_0210 OUTPUT.  
  SET PF-STATUS 'STATUS210'.  
  SET TITLEBAR 'TITLE210'.  
ENDMODULE.  
*\*&---------------------------------------------------------------------\**  
*\*&      Module  USER\_COMMAND\_0210  INPUT*  
*\*&---------------------------------------------------------------------\**  
*\*       text*  
*\*----------------------------------------------------------------------\**  
MODULE user\_command\_0210 INPUT.  
  
  CASE ok\_code.  
    WHEN 'BACK'.  
      LEAVE TO SCREEN 0.  
    WHEN 'EXIT'.  
      LEAVE PROGRAM.  
    WHEN 'FC03'.  
      DATA lv\_id2 TYPE i.  
      SELECT MAX( id\_pk )  
        FROM zorders  
        INTO lv\_id2.  
      gs\_zorders-id\_pk = lv\_id2 + 1 .  
  
      z\_cl\_second2=>create\_object( )->write\_to\_order( gs\_zorders ).  
  
    WHEN OTHERS.  
  ENDCASE.  
  
ENDMODULE.  
*\*&---------------------------------------------------------------------\**  
*\*&      Form  USE\_SCREEN\_ORDER*  
*\*&---------------------------------------------------------------------\**  
*\*       text*  
*\*----------------------------------------------------------------------\**  
*\*  -->  p1        text*  
*\*  <--  p2        text*  
*\*----------------------------------------------------------------------\**  
FORM use\_screen\_order .  
  
  CALL SCREEN 210.  
  
ENDFORM.

Методы родительского класса

Метод CREATE\_FCAT

 method CREATE\_FCAT.  
  
    CLEAR mt\_fcat.  
  
  CALL FUNCTION 'LVC\_FIELDCATALOG\_MERGE'  
    EXPORTING  
*\*     I\_BUFFER\_ACTIVE  =*  
      i\_structure\_name = mo\_data\_name2  
*\*     I\_CLIENT\_NEVER\_DISPLAY       = 'X'*  
*\*     I\_BYPASSING\_BUFFER           =*  
*\*     I\_INTERNAL\_TABNAME           =*  
    CHANGING  
      ct\_fieldcat      = mt\_fcat  
*\*     EXCEPTIONS*  
*\*     INCONSISTENT\_INTERFACE       = 1*  
*\*     PROGRAM\_ERROR    = 2*  
*\*     OTHERS           = 3*  
    .  
  
  
  
  LOOP AT mt\_fcat ASSIGNING FIELD-SYMBOL(<ls\_fcat>).  
    IF <ls\_fcat>-fieldname = 'SELL'.  
     <ls\_fcat>-checkbox = 'X'.  
    ENDIF.  
  
  ENDLOOP.  
  
  
  IF sy-subrc <> 0.  
*\* Implement suitable error handling here*  
  ENDIF.  
  
  endmethod.

Метод RUN

method RUN.  
  
     me->select( ).  
     me->write( ).  
  
  endmethod.

Метод SCREEN\_MODIFY

  method SCREEN\_MODIFY.  
     DATA: lv\_invisible   TYPE c,  
          lv\_active      TYPE c,  
          lv\_output      TYPE c,  
          lv\_input       TYPE c,  
          lv\_invisible\_n TYPE c,  
          lv\_active\_n    TYPE c,  
          lv\_output\_n    TYPE c,  
          lv\_input\_n     TYPE c.  
  
  
  
      lv\_invisible = 1.  
      lv\_active    = 0.  
      lv\_output    = 0.  
      lv\_input     = 0.  
      lv\_invisible\_n = 0.  
      lv\_active\_n    = 1.  
      lv\_output\_n    = 1.  
      lv\_input\_n     = 1.  
  
  
    LOOP AT SCREEN.  
      CASE screen-group1.  
        WHEN iv\_visible\_groupe.  
          screen-invisible = lv\_invisible.  
          screen-active    = lv\_active.  
          screen-output    = lv\_output.  
          screen-input     = lv\_input.  
          MODIFY SCREEN.  
        WHEN iv\_invisible\_groupe.  
          screen-invisible = lv\_invisible\_n.  
          screen-active    = lv\_active\_n.  
          screen-output    = lv\_output\_n.  
          screen-input     = lv\_input\_n.  
          MODIFY SCREEN.  
      ENDCASE.  
    ENDLOOP.  
  endmethod.

Метод SHOW\_ALV

 METHOD show\_alv.  
  
  
    DATA: mo\_data\_name2 TYPE dd02l-tabname.  
    ASSIGN mo\_data->\* to FIELD-SYMBOL(<mt\_att>).  
    ASSIGN mo\_data\_name->\* to FIELD-SYMBOL(<mo\_data\_name>).  
    WRITE <mo\_data\_name> to mo\_data\_name2.  
  
    IF go\_cont IS BOUND.  
    go\_cont->free( ).  
  ENDIF.  
  
  
      CREATE OBJECT go\_cont  
        EXPORTING  
          container\_name              = 'CONTAINER'  
        EXCEPTIONS  
          cntl\_error                  = 1  
          cntl\_system\_error           = 2  
          create\_error                = 3  
          lifetime\_error              = 4  
          lifetime\_dynpro\_dynpro\_link = 5  
          OTHERS                      = 6.  
      IF sy-subrc <> 0.  
        MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno  
                   WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.  
      ENDIF.  
     create\_alv( ).  
  
  
  
  ENDMETHOD.

Метод CREATE\_ALV

  METHOD create\_alv.  
  
    ASSIGN mo\_data->\* to FIELD-SYMBOL(<mt\_att>).  
    ASSIGN mo\_data\_name->\* to FIELD-SYMBOL(<mo\_data\_name>).  
    WRITE <mo\_data\_name> to mo\_data\_name2.  
  
  
  
    DATA: ls\_layout TYPE lvc\_s\_layo.  
  
    CREATE OBJECT go\_alv  
      EXPORTING  
        i\_parent          = go\_cont  
      EXCEPTIONS  
        error\_cntl\_create = 1  
        error\_cntl\_init   = 2  
        error\_cntl\_link   = 3  
        error\_dp\_create   = 4  
        OTHERS            = 5.  
    IF sy-subrc <> 0.  
      MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno  
                 WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.  
    ENDIF.  
  
    ls\_layout-stylefname = 'CELLTAB'.  
  
    CREATE OBJECT go\_event EXPORTING in\_alv = go\_alv mt\_auto2 = mo\_data mo\_auto = mo\_auto mv\_nametab = mo\_data\_name.  
  
  
    set\_event( go\_event ).  
  
    create\_fcat( ).  
  
    CALL METHOD go\_alv->set\_table\_for\_first\_display  
      EXPORTING  
*\*       i\_buffer\_active  =*  
*\*       i\_bypassing\_buffer            =*  
*\*       i\_consistency\_check           =*  
        i\_structure\_name = mo\_data\_name2  
*\*       is\_variant       =*  
*\*       i\_save           =*  
*\*       i\_default        = 'X'*  
*\*       is\_layout        =*  
*\*       is\_print         =*  
*\*       it\_special\_groups             =*  
*\*       it\_toolbar\_excluding          =*  
*\*       it\_hyperlink     =*  
*\*       it\_alv\_graphics  =*  
*\*       it\_except\_qinfo  =*  
*\*       ir\_salv\_adapter  =*  
      CHANGING  
        it\_outtab        = <mt\_att>  
       it\_fieldcatalog  = mt\_fcat  
*\*       it\_sort          =*  
*\*       it\_filter        =*  
*\*  EXCEPTIONS*  
*\*       invalid\_parameter\_combination = 1*  
*\*       program\_error    = 2*  
*\*       too\_many\_lines   = 3*  
*\*       others           = 4*  
      .  
    IF sy-subrc <> 0.  
*\* Implement suitable error handling here*  
    ENDIF.  
  ENDMETHOD.

Метод REFRESH\_ALV

  method REFRESH\_ALV.  
  
    CALL METHOD go\_alv->refresh\_table\_display  
*\*    EXPORTING*  
*\*      is\_stable      =*  
*\*      i\_soft\_refresh =*  
*\*    EXCEPTIONS*  
*\*      finished       = 1*  
*\*      others         = 2*  
          .  
  IF sy-subrc <> 0.  
*\*   Implement suitable error handling here*  
  ENDIF.  
  
  endmethod.

Метод SET\_EVENT

  METHOD set\_event.  
  
    SET HANDLER go\_event->on\_double\_click FOR go\_alv.  
    SET HANDLER go\_event->on\_toolbar FOR go\_alv.  
    SET HANDLER go\_event->user\_command FOR go\_alv.  
  
  ENDMETHOD.

Методы класса наследника

Метод SELECT

  METHOD select.  
*\*CALL METHOD SUPER->SELECT*  
*\*    .*  
  
  
  
    SELECT \*  
     FROM zauto  
     INTO TABLE mt\_att  
     WHERE model IN mr\_model AND brand IN mr\_brand.  
  
      GET REFERENCE OF mt\_att INTO mo\_data.  
      GET REFERENCE OF 'ZAUTO' INTO mo\_data\_name.  
  ENDMETHOD.

Метод SELECT2

  METHOD select2.  
*\*CALL METHOD SUPER->SELECT*  
*\*    .*  
  
     SELECT \*  
     FROM zorders  
     INTO TABLE mt\_att2  
       WHERE id\_auto\_fk = IV\_FK.  
  
  GET REFERENCE OF mt\_att2 INTO mo\_data.  
  GET REFERENCE OF 'ZORDERS' INTO mo\_data\_name.  
  
  z\_cl\_second2=>create\_object( )->show\_alv( ).  
  endmethod.

Метод WRITE

  method WRITE.  
*\*CALL METHOD SUPER->WRITE*  
*\*    .*  
    LOOP AT  MT\_ATT ASSIGNING FIELD-SYMBOL(<LS\_TABLE>).  
   WRITE: <LS\_TABLE>-MODEL, ' ' ,  
        <LS\_TABLE>-BRAND, ' ' ,  
        <LS\_TABLE>-amount, ' ' , /.  
  ENDLOOP.  
  endmethod.

Метод SET\_AUTO2

method SET\_AUTO2.  
  
      mr\_brand = rv\_brand.  
      mr\_model = rv\_model.  
      Mt\_att = rt\_auto.  
  
    ENDMETHOD.

Метод WRITE\_TO\_BD

METHOD write\_to\_bd.  
  DATA : ls\_table TYPE zauto.  
  select SINGLE \*  
    from zauto  
    INTO ls\_table  
    WHERE model = is\_transport-model AND brand = is\_transport-brand.  
  IF sy-subrc = 0.  
    MESSAGE 'ЗАПИСЬ УЖЕ СУЩЕСТВУЕТ' TYPE 'I'.  
  ELSE .  
    MODIFY zauto FROM is\_transport.  
    MESSAGE 'ЗАПИСЬ ЗАПИСАНА' TYPE 'I'.  
  ENDIF.  
  
  IF go\_cont IS BOUND.  
    go\_alv->refresh\_table\_display( ).  
    endif.  
  
  
  ENDMETHOD.

Метод CREATE\_OBJECT

  METHOD create\_object.  
  
    IF mo\_auto IS NOT BOUND.  
      CREATE OBJECT mo\_auto.  
    ENDIF.  
    ro\_auto = mo\_auto.  
  ENDMETHOD.

Метод UPDATE\_TABLE

  method UPDATE\_TABLE.  
  
    mt\_att = mt\_auto.  
  
  endmethod.

Метод WRITE\_TO\_ORDER

  method WRITE\_TO\_ORDER.  
  
    DATA : ls\_table TYPE zorders.  
  select SINGLE \*  
    from zorders  
    INTO ls\_table  
    WHERE id\_auto\_fk = is\_order-id\_auto\_fk AND first\_name = is\_order-first\_name AND second\_name = is\_order-second\_name.  
  IF sy-subrc = 0.  
    MESSAGE 'ЗАПИСЬ УЖЕ СУЩЕСТВУЕТ' TYPE 'I'.  
  ELSE .  
    MODIFY zorders FROM is\_order.  
    COMMIT WORK.  
    MESSAGE 'ЗАПИСЬ ЗАПИСАНА' TYPE 'I'.  
  ENDIF.  
  
  IF go\_cont IS BOUND.  
    go\_alv->refresh\_table\_display( ).  
    endif.  
  
  
  endmethod.

Метод SELECT\_ALL\_ORDER

  method SELECT\_ALL\_ORDER.  
  
    SELECT \*  
     FROM zorders  
     INTO TABLE mt\_att2  
       .  
  
  GET REFERENCE OF mt\_att2 INTO mo\_data.  
  GET REFERENCE OF 'ZORDERS' INTO mo\_data\_name.  
  
  z\_cl\_second2=>create\_object( )->show\_alv( ).  
  
  endmethod.

Метод SELECT\_ALL\_AUTO

  method SELECT\_ALL\_AUTO.  
  
  
    SELECT \*  
     FROM zauto  
     INTO TABLE mt\_att  
    .  
  
      GET REFERENCE OF mt\_att INTO mo\_data.  
      GET REFERENCE OF 'ZAUTO' INTO mo\_data\_name.  
  
   z\_cl\_second2=>create\_object( )->show\_alv( ).  
  
  endmethod.

Методы класса обработчика событий

Метод CONSTRUCTOR

  METHOD constructor.  
  
    mo\_alv = in\_alv.  
    mt\_auto = mt\_auto2.  
    go\_auto2 = mo\_auto.  
    mv\_tabname = mv\_nametab.  
  
  ENDMETHOD.

Метод ON\_DOUBLE\_CLICK

  METHOD on\_double\_click.  
  
    DATA: lv\_mess   TYPE string,  
          lv\_row    TYPE char10,  
          lv\_column TYPE char15,  
          lt\_table  TYPE zauto,  
          lv\_name   TYPE char20.  
  
    ASSIGN mv\_tabname->\* TO FIELD-SYMBOL(<lv\_name>).  
    WRITE <lv\_name> TO lv\_name.  
    WRITE e\_row-index TO lv\_row.  
    WRITE e\_column TO lv\_column.  
    IF lv\_name = 'ZAUTO'.  
      ASSIGN mt\_auto->\* TO FIELD-SYMBOL(<mt\_auto>).  
      MOVE <mt\_auto> TO mt\_table.  
      READ TABLE mt\_table INDEX e\_row-index INTO lt\_table.  
      z\_cl\_second2=>create\_object( )->select2( lt\_table-id\_pk ).  
    ENDIF.  
  
  ENDMETHOD.

Метод USER\_COMMAND

 METHOD user\_command.  
  
    DATA: lt\_mess TYPE string,  
          lt\_rows TYPE lvc\_t\_row,  
          lv\_name TYPE char20.  
  
    ASSIGN mv\_tabname->\* TO FIELD-SYMBOL(<lv\_name>).  
    WRITE <lv\_name> TO lv\_name.  
  
    mo\_alv->get\_selected\_rows( IMPORTING et\_index\_rows = lt\_rows ).  
  
    CASE e\_ucomm.  
      WHEN 'ADD'.  
        IF lv\_name = 'ZAUTO'.  
          PERFORM use\_screen IN PROGRAM z\_auto\_copy.  
        ELSE.  
          PERFORM use\_screen\_order IN PROGRAM z\_auto\_copy.  
        ENDIF.  
  
      WHEN 'DELETE'.  
  
        FIELD-SYMBOLS: <mt\_auto> TYPE STANDARD TABLE.  
        ASSIGN mt\_auto->\* TO <mt\_auto>.  
  
  
        mo\_alv->get\_selected\_rows( IMPORTING et\_index\_rows = lt\_rows ).  
  
        LOOP AT lt\_rows ASSIGNING FIELD-SYMBOL(<ls\_rows>).  
          READ TABLE <mt\_auto> ASSIGNING FIELD-SYMBOL(<ls\_auto>) INDEX <ls\_rows>-index.  
          IF sy-subrc = 0.  
            DELETE (lv\_name) FROM <ls\_auto>.  
            COMMIT WORK.  
          ENDIF.  
          DELETE <mt\_auto> INDEX <ls\_rows>-index.  
        ENDLOOP.  
  
        go\_auto2->update\_table( mt\_table ).  
  
        mo\_alv->refresh\_table\_display( ).  
      WHEN 'UPDATE'.  
        CASE lv\_name.  
          WHEN 'ZAUTO'.  
            go\_auto2->select\_all\_order( ).  
          WHEN 'ZORDERS'.  
            go\_auto2->select\_all\_auto( ).  
        ENDCASE.  
  
        mo\_alv->refresh\_table\_display( ).  
  
  
    ENDCASE.  
  
  ENDMETHOD.

Метод ON\_TOOLBAR

  METHOD on\_toolbar.  
  
    FIELD-SYMBOLS: <ls\_toolbar> LIKE LINE OF e\_object->mt\_toolbar.  
  
    APPEND INITIAL LINE TO e\_object->mt\_toolbar ASSIGNING <ls\_toolbar> .  
    <ls\_toolbar>-function = 'ADD'.  
    <ls\_toolbar>-icon = icon\_positive.  
    <ls\_toolbar>-quickinfo = 'Добавить строку'.  
    <ls\_toolbar>-text = 'Добавить строку'.  
  
    APPEND INITIAL LINE TO e\_object->mt\_toolbar ASSIGNING <ls\_toolbar> .  
    <ls\_toolbar>-function = 'DELETE'.  
    <ls\_toolbar>-icon = icon\_delete.  
    <ls\_toolbar>-quickinfo = 'Удалить строку'.  
    <ls\_toolbar>-text = 'Удалить строку'.  
  
    APPEND INITIAL LINE TO e\_object->mt\_toolbar ASSIGNING <ls\_toolbar> .  
    <ls\_toolbar>-function = 'UPDATE'.  
    <ls\_toolbar>-icon = icon\_refresh.  
    <ls\_toolbar>-quickinfo = 'Сменить таблицу'.  
    <ls\_toolbar>-text = 'Сменить таблицу'.  
  
  
  ENDMETHOD.