Бесплатная электронная книга

УЧУСЬ

ABAP

Free unaffiliated eBook created from
Stack Overflow contributors.

#abap

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: abap

It is an unofficial and free ABAP ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official ABAP.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с ABAP

замечания

ABAP - это язык программирования, разработанный SAP для программирования бизнес-приложений в среде SAP.

Ранее только процедурный, ABAP теперь также является объектно-ориентированным языком благодаря усовершенствованию объектов ABAP.

Версии

Версия	Дата выхода
ABAP 7.50	2015-10-20
ABAP 7.40	2012-11-29
ABAP 7.0	2006-04-01
ABAP 6.40	2004-04-01
ABAP 6.20	2002-04-01
ABAP 6.10	2001-07-01
ABAP 4.6C	2001-04-01
ABAP 4.6A	1999-12-01
ABAP 4.5	1999-03-01
ABAP 4.0	1998-06-01
ABAP 3.0	1997-02-20

Examples

Привет, мир

```
PROGRAM zhello_world.
START-OF-SELECTION.
    WRITE 'Hello, World!'.
```

Вместо того, чтобы печатать на консоли, ABAP записывает значения в список, который

будет отображаться, как только будет выполнена основная логика.

Hello World в объектах ABAP

```
PROGRAM zhello_world.

CLASS main DEFINITION FINAL CREATE PRIVATE.
  PUBLIC SECTION.
    CLASS-METHODS: start.
ENDCLASS.

CLASS main IMPLEMENTATION.
  METHOD start.
    cl_demo_output=>display( 'Hello World!' ).
  ENDMETHOD.
ENDCLASS.

START-OF-SELECTION.
  main=>start( ).
```

Прочитайте Начало работы с ABAP онлайн: https://riptutorial.com/ru/abap/topic/1196/начало-работы-с-abap

глава 2: ABAP GRID List Viewer (ALV)

Examples

Создание и отображение ALV

В этом примере `cl_salv_table` наиболее простое создание ALV с использованием класса `cl_salv_table` и дополнительных параметров форматирования. Дополнительные опции форматирования будут включены после блока `TRY ENDTRY` и перед `alv->display( )` метода `alv->display( )` .

Все последующие примеры с использованием подхода ABAP Objects к созданию ALV будут использовать этот пример в качестве отправной точки.

```
DATA: t_spfli       TYPE STANDARD TABLE OF spfli,
      alv           TYPE REF TO cl_salv_table,
      error_message TYPE REF TO cx_salv_msg.

" Fill the internal table with example data
SELECT * FROM spfli INTO TABLE t_spfli.

" Fill ALV object with data from the internal table
TRY.
    cl_salv_table=>factory(
      IMPORTING
        r_salv_table = alv
      CHANGING
        t_table      = t_spfli ).
  CATCH cx_salv_msg INTO error_message.
    " error handling
ENDTRY.

" Use the ALV object's display method to show the ALV on the screen
alv->display( ).
```

Оптимизировать ширину столбца ALV

В этом примере показано, как оптимизировать ширину столбца, чтобы заголовки столбцов и данные не прерывались.

```
alv->get_columns( )->set_optimize( ).
```

Скрыть столбцы в ALV

В этом примере скрывается поле `MANDT` (клиент) из ALV. Обратите внимание, что параметр, переданный в `get_column( )` *должен* быть капитализирован, чтобы это работало.

```
alv->get_columns( )->get_column( 'MANDT' )->set_visible( if_salv_c_bool_sap=>false ).
```

Переименование заголовков столбцов в ALV

Текст столбца может изменяться при горизонтальном изменении размера столбца. Для этого есть три способа:

Имя метода	Максимальная длина заголовка
`set_short_text`	10
`set_medium_text`	20
`set_long_text`	40

В следующем примере показано использование всех трех. Объект `column` объявляется и создается как ссылка на результат `alv->get_columns( )->get_column( 'DISTID' )` . Имя столбца *должно* быть заглавными буквами. Это значит, что эта цепочка методов вызывается только один раз в своем экземпляре, а не выполняется каждый раз при изменении заголовка столбца.

```
DATA column TYPE REF TO cl_salv_column.
column = alv->get_columns( )->get_column( 'DISTID' ).

column->set_short_text( 'Dist. Unit' ).
column->set_medium_text( 'Unit of Distance' ).
column->set_long_text( 'Mass Unit of Distance (kms, miles)' ).
```

Включить функциональность панели инструментов ALV

Следующий вызов метода позволяет использовать многие дополнительные функции, такие как сортировка, фильтрация и экспорт данных.

```
alv->get_functions( )->set_all( ).
```

Включение любой другой строки в ALV

Этот метод повышает удобочитаемость, предоставляя последовательные строки чередующихся оттенков фона.

```
alv->get_display_settings( )->set_striped_pattern( if_salv_c_bool_sap=>true ).
```

Установка заголовка отображаемого ALV

По умолчанию, когда отображается ALV, заголовок вверху - это просто имя программы. Этот метод позволяет пользователю установить заголовок до 70 символов. В следующем примере показано, как можно настроить динамический заголовок, отображающий

количество отображаемых записей.

```
alv->get_display_settings( )->set_list_header( |Flight Schedule – { lines( t_spfli ) }
records| ).
```

Прочитайте ABAP GRID List Viewer (ALV) онлайн:
https://riptutorial.com/ru/abap/topic/4660/abap-grid-list-viewer--alv-

глава 3: Loops

замечания

При переходе по внутренним таблицам обычно предпочтительнее использовать `ASSIGN` для символа поля, а не для цикла `INTO` в рабочей области. Присвоение символов поля просто обновляет их ссылку, чтобы указать на следующую строку внутренней таблицы во время каждой итерации, тогда как использование `INTO` приводит к тому, что строка таблицы будет скопирована в рабочую область, что может быть дорогостоящим для таблиц с длинными / большими.

Examples

Внутренний стол

```
LOOP AT itab INTO wa.
ENDLOOP.

FIELD-SYMBOLS <fs> LIKE LINE OF itab.
LOOP AT itab ASSIGNING <fs>.
ENDLOOP.

LOOP AT itab ASSIGNING FIELD-SYMBOL(<fs>).
ENDLOOP.

LOOP AT itab REFERENCE INTO dref.
ENDLOOP.

LOOP AT itab TRANSPORTING NO FIELDS.
ENDLOOP.
```

Условный цикл

Если в цикл должны быть взяты только строки, соответствующие определенному условию, добавление `WHERE` может быть добавлено.

```
LOOP AT itab INTO wa WHERE f1 = 'Max'.
ENDLOOP.
```

Пока цикл

ABAP также предлагает обычный `WHILE` -Loop, который выполняется до тех пор, пока данное выражение не станет равно false. Системный `sy-index` будет увеличен для каждого шага цикла.

```
WHILE condition.
```

```
* do something
ENDWHILE
```

Do Loop

Без какого-либо дополнения `DO` -Loop работает бесконечно или, по крайней мере, до тех пор, пока он не будет явно удален изнутри. Системный `sy-index` будет увеличен для каждого шага цикла.

```
DO.
* do something... get it?
* call EXIT somewhere
ENDDO.
```

Добавление `TIMES` предлагает очень удобный способ повторить код (`amount` представляет значение типа `i`).

```
DO amount TIMES.
* do several times
ENDDO.
```

Общие команды

Чтобы разбить петли, можно использовать команду `EXIT` .

```
DO.
    READ TABLE itab INDEX sy-index INTO DATA(wa).
    IF sy-subrc <> 0.
        EXIT. "Stop this loop if no element was found
    ENDIF.
    " some code
ENDDO.
```

Чтобы перейти к следующему шагу цикла, можно использовать команду `CONTINUE` .

```
DO.
    IF sy-index MOD 1 = 0.
        CONTINUE. " continue to next even index
    ENDIF.
    " some code
ENDDO.
```

`CHECK` является `CONTINUE` с условием. Если условие окажется **ложным** , будет выполнено `CONTINUE` . Другими словами: *цикл будет выполняться только с шагом, если условие истинно* .

Этот пример `CHECK` ...

```
DO.
    " some code
    CHECK sy-index < 10.
```

```
    " some code
ENDDO.
```

... эквивалентно ...

```
DO.
    " some code
    IF sy-index >= 10.
        CONTINUE.
    ENDIF.
    " some code
ENDDO.
```

Прочитайте Loops онлайн: https://riptutorial.com/ru/abap/topic/2270/loops

глава 4: Внутренние таблицы

Examples

Типы внутренних таблиц

```
DATA: <TABLE NAME> TYPE <SORTED|STANDARD|HASHED> TABLE OF <TYPE NAME>
      WITH <UNIQUE|NON-UNIQUE> KEY <FIELDS FOR KEY>.
```

Стандартная таблица

В этой таблице есть все записи, хранящиеся в линейном режиме, и записи доступны линейным способом. Для больших размеров таблиц доступ к таблице может быть медленным.

Сортированная таблица

Требуется дополнение `WITH UNIQUE | NON-UNIQUE KEY` . Поиск выполняется быстро из-за выполнения двоичного поиска. Записи не могут быть добавлены в эту таблицу, так как они могут нарушить порядок сортировки, поэтому они всегда вставляются с использованием ключевого слова `INSERT` .

Хешированный стол

Требуется дополнение `WITH UNIQUE | NON-UNIQUE KEY` . Использует собственный алгоритм хеширования для поддержки пар ключ-значение. Теоретические поиски могут быть такими же медленными, как и таблица `STANDARD` но практически они быстрее, чем таблица `SORTED` занимающая постоянное количество времени, независимо от размера таблицы.

Декларация внутренних таблиц ABAP

Внутренняя декларация таблицы на основе определения локального типа

```
" Declaration of type
TYPES: BEGIN OF ty_flightb,
         id        TYPE fl_id,
         dat       TYPE fl_date,
         seatno    TYPE fl_seatno,
         firstname TYPE fl_fname,
         lastname  TYPE fl_lname,
         fl_smoke  TYPE fl_smoker,
         classf    TYPE fl_class,
         classb    TYPE fl_class,
```

```
         classe    TYPE fl_class,
         meal      TYPE fl_meal,
         service   TYPE fl_service,
         discout   TYPE fl_discnt,
       END OF lty_flightb.

" Declaration of internal table
DATA t_flightb TYPE STANDARD TABLE OF ty_flightb.
```

Декларация на основе таблицы базы данных

```
DATA t_flightb TYPE STANDARD TABLE OF flightb.
```

Внутренняя декларация внутренней таблицы

Требуется версия ABAP> 7.4

```
TYPES t_itab TYPE STANDARD TABLE OF i WITH EMPTY KEY.

DATA(t_inline) = VALUE t_itab( ( 1 ) ( 2 ) ( 3 ) ).
```

Внутренняя таблица с объявлением строк заголовка

В ABAP есть таблицы со строками заголовков и таблицы без строк заголовка. Таблицы с заголовками являются более старой концепцией и не должны использоваться в новой разработке.

Внутренняя таблица: стандартная таблица с / без строки заголовка

Этот код объявляет таблицу `i_compc_all` существующей структурой `compc_str` .

```
DATA: i_compc_all TYPE STANDARD TABLE OF compc_str WITH HEADER LINE.
DATA: i_compc_all TYPE STANDARD TABLE OF compc_str.
```

Внутренняя таблица: Хешированная таблица с / без строки заголовка

```
DATA: i_map_rules_c TYPE HASHED TABLE OF /bic/ansdomm0100 WITH HEADER LINE
DATA: i_map_rules_c TYPE HASHED TABLE OF /bic/ansdomm0100
```

Объявление рабочей области для таблиц без заголовка

Рабочая область (обычно сокращенно *wa*) имеет ту же структуру, что и таблица, но может содержать только одну строку (WA - это структура таблицы только с одним измерением).

```
DATA: i_compc_all_line LIKE LINE OF i_compc_all.
```

Чтение, запись и ввод во внутренние таблицы

Чтение, запись и вставка во внутренние таблицы с помощью строки заголовка:

```
" Read from table with header (using a loop):
LOOP AT i_compc_all.              " Loop over table i_compc_all and assign header line
  CASE i_compc_all-ftype.         " Read cell ftype from header line from table i_compc_all
    WHEN 'B'.                     " Bill-to customer number transformation
      i_compc_bil = i_compc_all.  " Assign header line of table i_compc_bil with content of
header line i_compc_all
      APPEND i_compc_bil.         " Insert header line of table i_compc_bil into table
i_compc_bil
    " ... more WHENs
  ENDCASE.
ENDLOOP.
```

> Напоминание. Внутренние таблицы с заголовками запрещены в объектно-ориентированных контекстах. Рекомендуется всегда использовать внутренние таблицы *без* строк заголовка.

Чтение, запись и вставка во внутренние таблицы без строки заголовка:

```
" Loop over table i_compc_all and assign current line to structure i_compc_all_line
LOOP AT i_compc_all INTO i_compc_all_line.
  CASE i_compc_all_line-ftype.                " Read column ftype from current line (which as
assigned into i_compc_all_line)
    WHEN 'B'.                                 " Bill-to customer number transformation
      i_compc_bil_line = i_compc_all_line.    " Copy structure
      APPEND i_compc_bil_line TO i_compc_bil. " Append structure to table
    " more WHENs ...
  ENDCASE.
ENDLOOP.


" Insert into table with Header:
INSERT TABLE i_sap_knb1.                      " insert into TABLE WITH HEADER: insert table
header into it's content
insert i_sap_knb1_line into table i_sap_knb1. " insert into HASHED TABLE: insert structure
i_sap_knb1_line into hashed table i_sap_knb1
APPEND p_t_errorlog_line to p_t_errorlog.     " insert into STANDARD TABLE: insert structure /
wa p_t_errorlog_line into table p_t_errorlog_line
```

Прочитайте Внутренние таблицы онлайн: https://riptutorial.com/ru/abap/topic/1647/внутренние-таблицы

глава 5: Выражения управляющего потока

Examples

IF / ELSEIF / ELSE

```
IF lv_foo = 3.
  WRITE: / 'lv_foo is 3'.
ELSEIF lv_foo = 5.
  WRITE: / 'lv_foo is 5'.
ELSE.
  WRITE: / 'lv_foo is neither 3 nor 5'.
ENDIF.
```

ДЕЛО

```
CASE lv_foo.
  WHEN 1.
    WRITE: / 'lv_foo is 1'.
  WHEN 2.
    WRITE: / 'lv_foo is 2'.
  WHEN 3.
    WRITE: / 'lv_foo is 3'.
  WHEN OTHERS.
    WRITE: / 'lv_foo is something else'.
ENDCASE
```

ПРОВЕРЯТЬ

`CHECK` - это простой оператор, который оценивает логическое выражение и выходит из текущего блока обработки, если он является ложным.

```
METHOD do_something.
  CHECK iv_input IS NOT INITIAL. "Exits method immediately if iv_input is initial

  "The rest of the method is only executed if iv_input is not initial
ENDMETHOD.
```

ASSERT

`ASSERT` используется в чувствительных областях, где вы хотите быть абсолютно уверенными, что переменная имеет определенное значение. Если логическое условие после `ASSERT` оказывается ложным, генерируется исключение, `ASSERTION_FAILED` (`ASSERTION_FAILED`).

```
ASSERT 1 = 1. "No Problem - Program continues

ASSERT 1 = 2. "ERROR
```

COND / SWITCH

`SWITCH` и `COND` предлагают специальную форму условного программного потока. В отличие от `IF` и `CASE` , они представляют разные значения, основанные на выражении, а не на выполнении операторов. Вот почему они считаются функциональными.

COND

Всякий раз, когда нужно учитывать несколько условий, `COND` может выполнять эту работу. Синтаксис довольно прост:

```
COND <type>(
    WHEN <condition> THEN <value>
    ...
    [ ELSE <default> | throw <exception> ]
).
```

Примеры

```
" Set screen element active depending on radio button
screen-active = COND i(
    WHEN p_radio = abap_true THEN 1
    ELSE 0 " optional, because type 'i' defaults to zero
).

" Check how two operands are related to each other
" COND determines its type from rw_compare
rw_compare = COND #(
    WHEN op1 < op2 THEN 'LT'
    WHEN op1 = op2 THEN 'EQ'
    WHEN op1 > op2 THEN 'GT'
).
```

ВЫКЛЮЧАТЕЛЬ

`SWITCH` - это аккуратный инструмент для отображения значений, поскольку он проверяет только на равенство, поэтому в некоторых случаях он короче, чем `COND` . Если был предоставлен неожиданный ввод, можно также исключить исключение. Синтаксис немного отличается:

```
SWITCH <type>(
    <variable>
    WHEN <value> THEN <new_value>
    ...
    [ ELSE <default> | throw <exception> ]
).
```

Примеры

```
DATA(lw_language) = SWITCH string(
    sy-langu
    WHEN 'E' THEN 'English'
    WHEN 'D' THEN 'German'
    " ...
    ELSE THROW cx_sy_conversion_unknown_langu( )
).
```

Прочитайте Выражения управляющего потока онлайн:
https://riptutorial.com/ru/abap/topic/7289/выражения-управляющего-потока

глава 6: Динамическое программирование

Examples

Поле-символы

Символы поля эквивалентны ABAP для указателей, за исключением того, что полевые символы всегда разыменовываются (изменить фактический адрес в памяти невозможно).

декларация

Чтобы объявить полевой символ, необходимо использовать ключевое слово `FIELD-SYMBOLS` . Типы могут быть общими (`ANY [... TABLE]`) Для обработки самых разных переменных.

```
FIELD-SYMBOLS: <fs_line>     TYPE any,    "generic
               <fs_struct>   TYPE kna1.   "non-generic
```

Назначение

Полевые символы `unassigned` декларируются при объявлении, а это означает, что они ничего не указывают. Доступ к неназначенному полю-символу приведет к исключению и, если не показано, короткому дампу. Поэтому состояние должно быть проверено с помощью `IS ASSIGNED` :

```
IF <fs> IS ASSIGNED.
*... symbol is assigned
ENDIF.
```

Поскольку они являются только ссылками, реальные данные не могут храниться внутри. Таким образом, объявленный `DATA` необходим в каждом случае использования.

```
DATA: w_name  TYPE string VALUE `Max`,
      w_index TYPE i      VALUE 1.

FIELD-SYMBOLS <fs_name> TYPE any.

ASSIGN w_name TO <fs_name>. "<fs_name> now gets w_name
<fs_name> = 'Manni'.        "Changes to <fs_name> now also affect w_name

* As <fs_name> is generic, it can also be used for numbers

ASSIGN w_index TO <fs_name>. "<fs_name> now refers to w_index.
ADD 1 TO <fs_name>.          "w_index gets incremented by one
```

отменяете

Иногда может быть полезно сбросить полевой символ. Это можно сделать с помощью `UNASSIGN` .

```
UNASSIGN <fs>.
* Access on <fs> now leads to an exception again
```

Использование для внутренних таблиц

Полевые символы могут использоваться для изменения внутренних таблиц.

```
LOOP AT itab INTO DATA(wa).
* Only modifies wa_line
    wa-name1 = 'Max'.
ENDLOOP.

LOOP AT itab ASSIGNING FIELD-SYMBOL(<fs>).
* Directly refers to a line of itab and modifies its values
    <fs>-name1 = 'Max'.
ENDLOOP.
```

Внимание! Полевые символы остаются назначенными даже после выхода из цикла. Если вы хотите повторно использовать их безопасно, немедленно отмените их назначение.

Ссылки на данные

Существенным для ссылок на данные является добавление `REF TO` после `TYPE` .

Динамическое создание структур

Если тип структуры должен быть определен во время выполнения, мы можем определить нашу целевую структуру в качестве ссылки на `data` типового типа.

```
DATA wa TYPE REF TO data.
```

Чтобы `wa` тип, мы используем оператор `CREATE DATA` . `TYPE` добавления можно указать:

Ссылка:

```
CREATE DATA wa TYPE kna1
```

- *Статические проверки активны, поэтому невозможно создать неизвестный тип*

Название:

```
CREATE DATA wa TYPE (lw_name_as_string)
```

- *`lw_name_as_string` нужны, а `lw_name_as_string` содержит `lw_name_as_string` типов в виде строки.*
- *Если тип не найден, будет `CX_SY_CREATE_DATA_ERROR` исключение типа `CX_SY_CREATE_DATA_ERROR`*

Для создания динамически создаваемых типов может быть указано дополнение `HANDLE` . `HANDLE`

получает объект, который наследуется от `CL_ABAP_DATADESCR` .

```
CREATE DATA dref TYPE HANDLE obj
```

- `obj` может быть создан с использованием **R** un **T** ime **T** ype **S** ervices
- потому что `dref` все еще является datareference, он должен быть разыменован (`->*`), который будет использоваться как datacontainer (обычно это делается с помощью полевых символов)

Службы типа RunTime

Службы типа RunTime (*короткие:* **RTTS**) используются либо для:

- создание типов (создание типа RunTime, *короткий:* **RTTC**)
- анализирующие типы (идентификатор типа RunTime, *короткий:* **RTTI**)

Классы

```
CL_ABAP_TYPEDESCR
  |
  |--CL_ABAP_DATADESCR
  |    |
  |    |--CL_ABAP_ELEMDESCR
  |    |--CL_ABAP_REFDESCR
  |    |--CL_ABAP_COMPLEXDESCR
  |        |
  |        |--CL_ABAP_STRUCTDESCR
  |        |--CL_ABAP_TABLEDESCR
  |
  |--CL_ABAP_OBJECTDESCR
     |
     |--CL_ABAP_CLASSDESCR
     |--CL_ABAP_INTFDESCR
```

`CL_ABAP_TYPEDESCR` - базовый класс. Он реализует необходимые методы для описания:

- `DESCRIBE_BY_DATA`
- `DESCRIBE_BY_NAME`
- `DESCRIBE_BY_OBJECT_REF`
- `DESCRIBE_BY_DATA_REF`

Прочитайте Динамическое программирование онлайн:
https://riptutorial.com/ru/abap/topic/4442/динамическое-программирование

глава 7: Единичное тестирование

Examples

Структура тестового класса

Тестовые классы создаются в качестве локальных классов в специальном тестировании.

Это базовая структура тестового класса:

```
CLASS lcl_test DEFINITION
               FOR TESTING
               DURATION SHORT
               RISK LEVEL HARMLESS.

PRIVATE SECTION.
  DATA:
    mo_cut TYPE REF TO zcl_dummy.

  METHODS:
    setup,

  "********* 30 chars *********|
  dummy_test                    for testing.
ENDCLASS.

CLASS lcl_test IMPLEMENTATION.
  METHOD setup.
    CREATE OBJECT mo_cut.
  ENDMETHOD.

  METHOD dummy_test.
    cl_aunit_assert=>fail( ).
  ENDMETHOD.
ENDCLASS.
```

Любой метод, объявленный с помощью `FOR TESTING` будет модульным тестом. `setup` - это специальный метод, который выполняется перед каждым тестом.

Отдельный доступ к данным из логики

Важным принципом модульного тестирования является разделение доступа к данным с бизнес-логикой. Одним из эффективных методов для этого является определение интерфейсов для доступа к данным. Ваш основной класс всегда использует ссылку на этот интерфейс вместо прямого чтения или записи данных.

в производственном коде основному классу будет предоставлен объект, который обертывает фактический доступ к данным. Это может быть оператор выбора, вызов функции mudule, что-то действительно. Важная часть состоит в том, что этот класс не должен выполнять ничего другого. Нет логики.

При тестировании основного класса вы даете ему объект, который вместо этого выполняет статические, поддельные данные.

Пример доступа к таблице SCARR

Интерфейс доступа к данным `ZIF_DB_SCARR`:

```
INTERFACE zif_db_scarr
  PUBLIC.
    METHODS get_all
      RETURNING
        VALUE(rt_scarr) TYPE scarr_tab .
ENDINTERFACE.
```

Поддельный класс данных и тестовый класс:

```
CLASS lcl_db_scarr DEFINITION.
  PUBLIC SECTION.
    INTERFACES: zif_db_scarr.
ENDCLASS.

CLASS lcl_db_scarr IMPLEMENTATION.
  METHOD zif_db_scarr~get_all.
    " generate static data here
  ENDMETHOD.
ENDCLASS.

CLASS lcl_test DEFINITION
  FOR TESTING
  DURATION SHORT
  RISK LEVEL HARMLESS.

  PRIVATE SECTION.
    DATA:
      mo_cut TYPE REF TO zcl_main_class.

    METHODS:
      setup.
ENDCLASS.

CLASS lcl_test IMPLEMENTATION.
  METHOD setup.
    DATA: lo_db_scarr TYPE REF TO lcl_db_scarr.

    CREATE OBJECT lo_db_scarr.

    CREATE OBJECT mo_cut
      EXPORTING
        io_db_scarr = lo_db_scarr.
  ENDMETHOD.
ENDCLASS.
```

Идея здесь заключается в том, что в производственном коде `ZCL_MAIN_CLASS` получит объект `ZIF_DB_SCARR` который выполняет `SELECT` и возвращает всю таблицу, в то время как единичный тест работает со статическим набором данных, определенным здесь, в модульном тесте.

Прочитайте Единичное тестирование онлайн: https://riptutorial.com/ru/abap/topic/3999/единичное-тестирование

глава 8: Классы сообщений / ключевое слово MESSAGE

Вступление

Оператор `MESSAGE` может использоваться для прерывания потока программы для отображения коротких сообщений пользователю. Содержимое сообщений может быть определено в коде программы, в текстовых символах программы или в отдельном классе сообщений, определенном в `SE91` .

замечания

Максимальная длина сообщения, включая параметры, переданные ему с помощью `&` , составляет 72 символа.

Examples

Определение класса сообщений

```
PROGRAM zprogram MESSAGE-ID sabapdemos.
```

Системное сообщение может быть сохранено в классе сообщений. Маркер `MESSAGE-ID` определяет класс сообщений `sabapdemos` для всей программы. Если это не используется, класс сообщения должен указываться при каждом вызове `MESSAGE` .

СООБЩЕНИЕ с предопределенным текстовым символом

```
PROGRAM zprogram MESSAGE-ID za.
...
MESSAGE i000 WITH TEXT-i00.
```

Сообщение отобразит текст, сохраненный в текстовом символе `i00` пользователю. Поскольку тип сообщения - i (как видно на `i000`), после выхода пользователя из диалогового окна поток программы будет продолжаться с точки вызова `MESSAGE` .

Хотя текст не пришел из класса сообщений `za` , необходимо указать `MESSAGE-ID` .

Сообщение без предопределенного класса сообщений

```
PROGRAM zprogram.
...
MESSAGE i050(sabapdemos).
```

Может быть неудобно определять класс сообщений для всей программы, поэтому можно определить класс сообщения, из которого сообщение приходит в самой инструкции `MESSAGE` . В этом примере будет отображаться сообщение `050` из класса сообщений `sabapdemos` .

Динамические сообщения

```
DATA: msgid TYPE sy-msgid VALUE 'SABAPDEMOS',
      msgty TYPE sy-msgty VALUE 'I',
      msgno TYPE sy-msgno VALUE '050'.

MESSAGE ID mid TYPE mtype NUMBER num.
```

Вышеупомянутый вызов `MESSAGE` является синонимом вызова `MESSAGE i050(sapdemos).` ,

Передача параметров сообщениям

Символ `&` может использоваться в сообщении, чтобы позволить ему передавать параметры.

Упорядоченные параметры

Сообщение `777` класса `sabapdemos` :

```
Message with type &1 &2 in event &3
```

Вызов этого сообщения тремя параметрами возвращает сообщение с использованием параметров:

```
MESSAGE i050(sabapdemos) WITH 'E' '010' 'START-OF-SELECTION`.
```

Это сообщение будет отображаться как `Message with type E 010 in event START-OF-SELECTION` . Число рядом с символом `&` обозначает порядок отображения параметров.

Неупорядоченные параметры

Сообщение `888` класса `sabapdemos` :

```
& & & &
```

Вызов этого сообщения аналогичен:

```
MESSAGE i050(sabapdemos) WITH 'param1' 'param2' 'param3' 'param4'.
```

Это вызовет `param1 param2 param3 param4` .

Прочитайте Классы сообщений / ключевое слово MESSAGE онлайн:

https://riptutorial.com/ru/abap/topic/10691/классы-сообщений---ключевое-слово-message

глава 9: Комментарии

Examples

Конец линии

Любой текст после " символа на одной линии закомментирована:

```
DATA ls_booking TYPE flightb. " Commented text
```

Полная линия

Символ * замечает всю строку. * Должен быть первым символом в строке.

```
* DATA ls_booking TYPE flightb. Nothing on this line will be executed.
```

Прочитайте Комментарии онлайн: https://riptutorial.com/ru/abap/topic/1644/комментарии

глава 10: Объекты ABAP

Examples

Объявление класса

Классы ABAP могут быть объявлены глобально или локально . Глобальный класс может использоваться любым объектом в репозитории ABAP. Напротив, локальный класс может использоваться только в пределах области, которую он объявляет.

```
CLASS lcl_abap_class DEFINITION.
  PUBLIC SECTION.
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.

CLASS lcl_abap_class IMPLEMENTATION.
ENDCLASS.
```

Конструктор, методы

Реализация класса:

```
CLASS lcl_abap_class DEFINITION.
  PUBLIC SECTION.
    METHODS: constructor,
             method1.
  PROTECTED SECTION.
  PRIVATE SECTION.
    METHODS: method2,
             method3.
ENDCLASS.

CLASS lcl_abap_class IMPLEMENTATION.
    METHOD constructor.
        "Logic
    ENDMETHOD.

    METHOD method1.
        "Logic
    ENDMETHOD.

    METHOD method2.
        "Logic
        method3( ).
    ENDMETHOD.
```

```
    METHOD method3.
        "Logic
    ENDMETHOD.
ENDCLASS.
```

Пример вызова метода:

```
DATA lo_abap_class TYPE REF TO lcl_abap_class.
CREATE OBJECT lo_abap_class. "Constructor call
lo_abap_class->method1( ).
```

Метод с параметрами (импорт, изменение, экспорт)

Реализация класса:

```
CLASS lcl_abap_class DEFINITION.
  PRIVATE SECTION.
    METHODS method1 IMPORTING iv_string TYPE string
                      CHANGING cv_string TYPE string
                     EXPORTING ev_string TYPE string.
ENDCLASS.

CLASS lcl_abap_class IMPLEMENTATION.
    METHOD method1.
        cv_string = iv_string.
        ev_string = 'example'.
    ENDMETHOD.
ENDCLASS.
```

Пример вызова метода:

```
method1 (
  EXPORTING iv_string = lv_string
  IMPORTING ev_string = lv_string2
   CHANGING cv_string = lv_string3
).
```

Метод с возвращаемым параметром

Реализация класса:

```
CLASS lcl_abap_class DEFINITION.
  PRIVATE SECTION.
    METHODS method1 RETURNING VALUE(rv_string) TYPE string.
ENDCLASS.

CLASS lcl_abap_class IMPLEMENTATION.
    METHOD method1.
        rv_string = 'returned value'.
    ENDMETHOD.
ENDCLASS.
```

Пример вызова метода:

```
lv_string = method1( ).
```

Обратите внимание, что параметры, объявленные с помощью `RETURNING` , передаются только по значению.

Наследование - определение

Информация

Наследование позволяет вывести новый класс из существующего класса. Вы делаете это, используя добавление **INHERITING FROM** в

Подкласс **CLASS ОПРЕДЕЛЕНИЕ ВНУТРЕННИЙ ОТ** суперкласса.

заявление. Подкласс нового класса наследует все компоненты существующего класса суперкласса. Новый класс называется подклассом класса, из которого он получен. Первоначальный класс называется суперклассом нового класса. Класс может иметь более одного прямого подкласса, но он может иметь только один прямой суперкласс.

Внедрение класса

```
CLASS lcl_vehicle DEFINITION.
ENDCLASS.

CLASS lcl_vehicle IMPLEMENTATION.
ENDCLASS.

CLASS lcl_car DEFINITION INHERITING FROM lcl_vehicle.
ENDCLASS.

CLASS lcl_car IMPLEMENTATION.
ENDCLASS.
```

Наследование - абстрактные и конечные методы и классы

Информация

АБСТРАКТНЫЕ и **ЗАКЛЮЧИТЕЛЬНЫЕ** дополнения к операторам **METHODS** и **CLASS** позволяют вам определять абстрактные и окончательные методы или классы.

Абстрактный метод определен в абстрактном классе и не может быть реализован в этом классе. Вместо этого он реализуется в подклассе класса.

Абстрактные классы не могут быть созданы.

Окончательный метод не может быть переопределен в подклассе. Заключительные классы не могут иметь подклассы. Они заключают дерево наследования.

Реализация класса:

```
CLASS lcl_abstract DEFINITION ABSTRACT.
    PUBLIC SECTION.
        METHODS: abstract_method ABSTRACT,
                 final_method FINAL
                 normal_method.

ENDCLASS.

CLASS lcl_abstract IMPLEMENTATION.
    METHOD final_method.
        "This method can't be redefined in child class!
    ENDMETHOD.

    METHOD normal_method.
        "Some logic
    ENDMETHOD.

        "We can't implement abstract_method here!

ENDCLASS.

CLASS lcl_abap_class DEFINITION INHERITING FROM lcl_abstract.
    PUBLIC SECTION.
        METHODS: abstract_method REDEFINITION,
                 abap_class_method.
ENDCLASS.

CLASS lcl_abap_class IMPLEMENTATION.
    METHOD abstract_method.
        "Abstract method implementation
    ENDMETHOD.

    METHOD abap_class_method.
        "Logic
    ENDMETHOD.
ENDCLASS.
```

Пример вызова метода:

```
DATA lo_class TYPE REF TO lcl_abap_class.
CREATE OBJECT lo_class.

lo_class->abstract_method( ).
lo_class->normal_method( ).
lo_class->abap_class_method( ).
lo_class->final_method( ).
```

Прочитайте Объекты ABAP онлайн: https://riptutorial.com/ru/abap/topic/2244/объекты-abap

глава 11: Объявление данных

Examples

Декларация встроенных данных

В определенных ситуациях декларации данных могут выполняться встроенными.

```
LOOP AT lt_sflight INTO DATA(ls_sflight).
    WRITE ls_sflight-carrid.
ENDLOOP.
```

Единая переменная

```
DATA begda TYPE sy-datum.
```

Декларация множественной переменной

```
DATA: begda TYPE sy-datum,
      endda TYPE sy-datum.
```

Декларация встроенных данных в инструкции SELECT

При использовании встроенной декларации данных внутри блока `SELECT...ENDSELECT` или `SELECT SINGLE` символ `@` должен использоваться как escape-символ для выражения `DATA(lv_cityto)` . После того, как используется escape-символ, все остальные переменные хоста также должны быть экранированы (как в случае с `lv_carrid` ниже).

```
DATA lv_carrid TYPE s_carr_id VALUE 'LH'.
SELECT SINGLE cityto FROM spfli
       INTO @DATA(lv_cityto)
       WHERE carrid = @lv_carrid
       AND   connid = 2402.
WRITE: / lv_cityto.
```

Выходы `BERLIN` .

Варианты переменных переменных

Различные типы переменных могут быть объявлены специальными опциями.

```
DATA: lv_string   TYPE string, " standard declaration
      lv_char     TYPE c,      " declares a character variable of length 1
      lv_char5(5) TYPE c,      " declares a character variable of length 5
      l_packed TYPE p LENGTH 10 DECIMALS 5 VALUE '1234567890.123456789'. " evaluates to
```

```
1,234,567,890.12346
```

Прочитайте Объявление данных онлайн: https://riptutorial.com/ru/abap/topic/1646/объявление-данных

глава 12: Открыть SQL

Examples

Оператор SELECT

SELECT - это оператор Open-SQL для чтения данных из одной или нескольких таблиц базы данных в объекты данных .

1. Выбор всех записей

```
* This returns all records into internal table lt_mara.
SELECT * FROM mara
         INTO lt_mara.
```

2. Выбор одиночной записи

```
* This returns single record if table consists multiple records with same key.
SELECT SINGLE * INTO TABLE lt_mara
                FROM mara
                WHERE matnr EQ '400-500'.
```

3. Выбор отдельных записей

```
* This returns records with distinct values.
SELECT DISTINCT * FROM mara
                  INTO TABLE lt_mara
                  ORDER BY matnr.
```

4. Совокупные функции

```
* This puts the number of records present in table MARA into the variable lv_var
SELECT COUNT( * ) FROM mara
                  INTO lv_var.
```

Прочитайте Открыть SQL онлайн: https://riptutorial.com/ru/abap/topic/6885/открыть-sql

глава 13: Регулярные выражения

Examples

Замена

Оператор `REPLACE` может работать с регулярными выражениями напрямую:

```
DATA(lv_test) = 'The quick brown fox'.
REPLACE ALL OCCURRENCES OF REGEX '\wo' IN lv_test WITH 'XX'.
```

Переменная `lv_test` будет оцениваться в `The quick bXXwn XXx` .

поиск

Оператор `FIND` может работать с регулярными выражениями напрямую:

```
DATA(lv_test) = 'The quick brown fox'.

FIND REGEX '..ck' IN lv_test.
" sy-subrc == 0

FIND REGEX 'a[sdf]g' IN lv_test.
" sy-subrc == 4
```

Объектно-ориентированные регулярные выражения

Для более сложных операций с регулярными выражениями лучше использовать `CL_ABAP_REGEX` и связанные с ним классы.

```
DATA: lv_test  TYPE string,
      lo_regex TYPE REF TO cl_abap_regex.

lv_test = 'The quick brown fox'.
CREATE OBJECT lo_regex
  EXPORTING
    pattern = 'q(...)\w'.

DATA(lo_matcher) = lo_regex->create_matcher( text = lv_test ).
WRITE: / lo_matcher->find_next( ).      " X
WRITE: / lo_matcher->get_submatch( 1 ). " uic
WRITE: / lo_matcher->get_offset( ).     " 4
```

Оценка регулярных выражений с помощью функции предиката

Функция сказуемое `matches` могут быть использованы для оценки строк на лету , без использования каких - либо деклараций объектов.

```
IF matches( val = 'Not a hex string'
            regex = '[0-9a-f]*' ).
  cl_demo_output=>display( 'This will not display' ).
ELSEIF matches( val = '6c6f7665'
                regex = '[0-9a-f]*' ).
  cl_demo_output=>display( 'This will display' ).
ENDIF.
```

Получение SubMatches с использованием OO-регулярных выражений

Используя метод `GET_SUBMATCH` класса `CL_ABAP_MATCHER` , мы можем получить данные в группах / подгруппах.

Цель: получить маркер справа от ключевого слова «Тип».

```
DATA: lv_pattern TYPE string VALUE 'type\s+(\w+)',
      lv_test TYPE string VALUE 'data lwa type mara'.

CREATE OBJECT ref_regex
  EXPORTING
        pattern     = lv_pattern
        ignore_case = c_true.

ref_regex->create_matcher(
    EXPORTING
        text    = lv_test
    RECEIVING
        matcher = ref_matcher
       ).

ref_matcher->get_submatch(
            EXPORTING
                 index = 0
            RECEIVING
                 submatch = lv_smatch.
```

Полученная переменная `lv_smatch` содержит значение `MARA` .

Прочитайте Регулярные выражения онлайн: https://riptutorial.com/ru/abap/topic/5113/регулярные-выражения

глава 14: Соглашения об именах

Синтаксис

- Символы, числа и _ могут использоваться для имени переменной.
- Два символа, использующие переменное состояние и тип объекта.
- Локальные переменные начинаются с L.
- Глобальные переменные начинаются с G.
- Параметр ввода функции начинается с I (импорт).
- Параметр выхода функции начинается с E (экспорт).
- Символом структур является S.
- Символом таблицы является T.

Examples

Локальное значение

```
data: lv_temp type string.
data: ls_temp type sy.
data: lt_temp type table of sy.
```

Глобальная переменная

```
data: gv_temp type string.
data: gs_temp type sy.
data: gt_temp type table of sy.
```

Прочитайте Соглашения об именах онлайн: https://riptutorial.com/ru/abap/topic/6770/соглашения-об-именах

глава 15: Струны

Examples

литералы

ABAP предлагает три разных оператора для объявления строковых или char-подобных переменных

Символы	Внутренний тип	длина	название
'...'	C	1-255 Chars	текстовые полевые литералы
`...`	CString	0-255 символов	текстовые строковые литералы
| ... |	CString	0-255 символов	шаблонные литералы

Обратите внимание, что диапазон длины применяется только к жестко закодированным значениям. Внутренне переменные `CString` имеют произвольную длину, а переменные типа `C` всегда имеют фиксированную длину.

Строковые шаблоны

Строковые шаблоны - это удобный способ смешивания литералов с значениями из переменных:

```
WRITE |Hello, { lv_name }, nice to meet you!|.
```

Он также может форматировать такие вещи, как даты. Чтобы использовать формат даты входа в систему:

```
WRITE |The order was completed on { lv_date DATE = USER } and can not be changed|.
```

Поддерживаются вызовы функциональных методов и выражения:

```
WRITE |Your token is { to_upper( lv_token ) }|.
WRITE |Version is: { cond #( when lv_date < sy-datum then 'out of date' else 'up to date' )
}|.
```

> **Внимание!** Прямое внедрение временных результатов (например, методов-вызовов) внутри шаблонов строк может привести к серьезным проблемам с производительностью (подробнее об этом читайте здесь). Хотя использование его внутри редко исполняемых утверждений в порядке, это заставляет вашу

программу быстро замедляться в циклах.

Конкатенация строк

Строковые и char-подобные переменные могут быть объединены с помощью команды ABAP `CONCATENATE` . Требуется дополнительная переменная для хранения результатов.

Пример:

```
CONCATENATE var1 var2 var3 INTO result.
"result now contains the values of var1, var2 & var3 stringed together without spaces
```

стенография

Более поздние версии ABAP предлагают очень короткий вариант конкатенации с использованием && (оператор цепочки).

```
DATA(lw_result) = `Sum: ` && lw_sum.
```

> **Внимание!** Стоит отметить, что использование временных результатов в сочетании с оператором Chaining внутри петель может привести к огромным проблемам с производительностью из-за растущих инструкций по копированию (подробнее об этом здесь).

Прочитайте Струны онлайн: https://riptutorial.com/ru/abap/topic/3531/струны

глава 16: Шаблоны программ

Синтаксис

- ОПРЕДЕЛЕНИЕ КЛАССА АБСТРАКТНЫЙ ЗАКЛЮЧИТЕЛЬНЫЙ делает класс программы по существу статичным, поскольку методы экземпляра никогда не могут использоваться. Цель состоит в том, чтобы сохранить класс минимальным.

Examples

Программа ОО с использованием основных методов мероприятия

```
REPORT z_template.

CLASS lcl_program DEFINITION ABSTRACT FINAL.

  PUBLIC SECTION.

    CLASS-METHODS start_of_selection.
    CLASS-METHODS initialization.
    CLASS-METHODS end_of_selection.

ENDCLASS.

CLASS lcl_program IMPLEMENTATION.

  METHOD initialization.

  ENDMETHOD.

  METHOD start_of_selection.

  ENDMETHOD.

  METHOD end_of_selection.

  ENDMETHOD.

ENDCLASS.

INITIALIZATION.

  lcl_program=>initialization( ).

START-OF-SELECTION.

  lcl_program=>start_of_selection( ).

END-OF-SELECTION.

  lcl_program=>end_of_selection( ).
```

Прочитайте Шаблоны программ онлайн: https://riptutorial.com/ru/abap/topic/10552/шаблоны-

программ

кредиты

S. No	Главы	Contributors
1	Начало работы с ABAP	Christian, Community, gkubed, Jagger, mkysoft
2	ABAP GRID List Viewer (ALV)	Achuth hadnoor, gkubed
3	Loops	Christian, Community, gkubed, Stu G
4	Внутренние таблицы	Community, gkubed, Michał Majer, Rahul Kadukar, Thorsten Niehues
5	Выражения управляющего потока	Community, gkubed, maillard
6	Динамическое программирование	Community, gkubed
7	Единичное тестирование	maillard
8	Классы сообщений / ключевое слово MESSAGE	gkubed
9	Комментарии	4444, Christian, gkubed
10	Объекты ABAP	Community, Michał Majer, Thomas Matecki
11	Объявление данных	Christian, gkubed
12	Открыть SQL	AKHIL RAJ, gkubed
13	Регулярные выражения	AKHIL RAJ, gkubed, maillard
14	Соглашения об именах	mkysoft
15	Струны	Achuth hadnoor, Community, maillard, nexxus, Suncatcher

16	Шаблоны программ	nath