EBook Gratuito

APPRENDIMENTO ABAP

Free unaffiliated eBook created from
Stack Overflow contributors.

#abap

Sommario

Di

Capitolo 1: Iniziare con ABAP

Osservazioni

ABAP è un linguaggio di programmazione sviluppato da SAP per la programmazione di applicazioni aziendali in ambiente SAP.

Precedentemente solo procedurale, ABAP ora è anche un linguaggio orientato agli oggetti grazie al miglioramento degli oggetti ABAP.

Versioni

Versione	Data di rilascio
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

Ciao mondo

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

Invece di stampare sulla console, ABAP scrive i valori in una lista che verrà visualizzata non appena viene eseguita la logica principale.

Ciao mondo in oggetti 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( ).
```

Leggi Iniziare con ABAP online: https://riptutorial.com/it/abap/topic/1196/iniziare-con-abap

Capitolo 2: ABAP GRID List Viewer (ALV)

Examples

Creazione e visualizzazione di un ALV

Questo esempio illustra la più semplice creazione di ALV utilizzando la classe `cl_salv_table` e senza ulteriori opzioni di formattazione. Ulteriori opzioni di formattazione sarebbero incluse dopo il blocco `TRY ENDTRY` e prima della chiamata al metodo `alv->display( ) .`

Tutti gli esempi successivi che utilizzano l'approccio Oggetti ABAP alla creazione di ALV utilizzeranno questo esempio come punto di partenza.

```
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( ).
```

Ottimizza larghezza colonna ALV

Questo esempio mostra come ottimizzare la larghezza della colonna in modo che le intestazioni di colonna e i dati non vengano troncati.

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

Nascondi colonne in un ALV

Questo esempio nasconde il campo `MANDT` (client) `MANDT` . Si noti che il parametro passato a `get_column( )` *deve* essere in maiuscolo affinché funzioni.

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

Rinominare le intestazioni delle colonne in un ALV

Il testo della colonna può cambiare sul ridimensionamento orizzontale di una colonna. Ci sono tre metodi per realizzare questo:

Nome del metodo	Lunghezza massima della direzione
`set_short_text`	10
`set_medium_text`	20
`set_long_text`	40

L'esempio seguente mostra l'utilizzo di tutti e tre. Un oggetto `column` viene dichiarato e istanziato come riferimento al risultato di `alv->get_columns( )->get_column( 'DISTID' )` . Il nome della colonna *deve* essere in lettere maiuscole. Questo è così che questo metodo di concatenamento viene chiamato una sola volta nella sua istanziazione, invece di essere eseguito ogni volta che viene modificata un'intestazione di colonna.

```
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)' ).
```

Abilita funzionalità barra degli strumenti ALV

La seguente chiamata di metodo consente l'utilizzo di molte funzionalità avanzate come l'ordinamento, il filtraggio e l'esportazione dei dati.

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

Abilitazione di ogni altro rigo di riga in ALV

Questo metodo aumenta la leggibilità dando righe consecutive alternando l'ombreggiatura del colore di sfondo.

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

Impostazione del titolo di un ALV visualizzato

Per impostazione predefinita, quando viene visualizzato un ALV, il titolo in alto è solo il nome del programma. Questo metodo consente all'utente di impostare un titolo di massimo 70 caratteri. L'esempio seguente mostra come è possibile impostare un titolo dinamico che mostri il numero di record visualizzati.

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

Leggi ABAP GRID List Viewer (ALV) online: https://riptutorial.com/it/abap/topic/4660/abap-grid-list-viewer--alv-

Capitolo 3: Apri SQL

Examples

SELECT statement

SELECT è un'istruzione Open-SQL per la lettura di dati da una o più tabelle di database in oggetti dati .

1. Selezione di tutti i record

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

2. Selezione di un singolo record

```
* 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. Selezione di record distinti

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

4. Funzioni aggregate

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

Leggi Apri SQL online: https://riptutorial.com/it/abap/topic/6885/apri-sql

Capitolo 4: Commenti

Examples

Fine della linea

Qualsiasi testo che segue un `"` carattere sulla stessa riga è commentato:

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

Linea completa

Il carattere `*` commenta un'intera riga. `*` Deve essere il primo carattere della riga.

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

Leggi Commenti online: https://riptutorial.com/it/abap/topic/1644/commenti

Capitolo 5: Controllo delle dichiarazioni di flusso

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.
```

ASTUCCIO

```
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
```

DAI UN'OCCHIATA

`CHECK` è una semplice istruzione che valuta un'espressione logica ed esce dal blocco di elaborazione corrente se è falso.

```
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.
```

AFFERMARE

`ASSERT` viene utilizzato in aree sensibili in cui si desidera essere assolutamente sicuri che una variabile abbia un valore specifico. Se la condizione logica dopo `ASSERT` risulta essere falsa, viene generata un'eccezione non `ASSERTION_FAILED` (`ASSERTION_FAILED`).

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

ASSERT 1 = 2. "ERROR
```

`SWITCH` e `COND` offrono una forma speciale di flusso del programma condizionale. A differenza di `IF` e `CASE` , rappresentavano valori diversi in base a un'espressione anziché a istruzioni di esecuzione. Ecco perché contano come funzionali.

COND

Ogni volta che devono essere considerate condizioni multiple, `COND` può fare il lavoro. La sintassi è abbastanza semplice:

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

Esempi

```
" 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'
).
```

INTERRUTTORE

`SWITCH` è uno strumento pulito per la mappatura dei valori, poiché controlla solo l'uguaglianza, in alcuni casi è più breve di `COND` . Se è stato fornito un input imprevisto, è anche possibile generare un'eccezione. La sintassi è leggermente diversa:

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

Esempi

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

Leggi Controllo delle dichiarazioni di flusso online:
https://riptutorial.com/it/abap/topic/7289/controllo-delle-dichiarazioni-di-flusso

Capitolo 6: Convenzioni di denominazione

Sintassi

- Caratteri, numeri e _ possono essere utilizzati per il nome della variabile.
- Due caratteri usando per lo stato variabile e il tipo di oggetto.
- Le variabili locali iniziano con L.
- Le variabili globali iniziano con G.
- Il parametro di input funzione inizia con I (importazione).
- Il parametro di uscita della funzione inizia con E (esportazione).
- Il simbolo delle strutture è S.
- Il simbolo della tabella è T.

Examples

Variabile locale

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

Variabile globale

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

Leggi Convenzioni di denominazione online: https://riptutorial.com/it/abap/topic/6770/convenzioni-di-denominazione

Capitolo 7: Dichiarazione dei dati

Examples

Dichiarazione dei dati in linea

In determinate situazioni, le dichiarazioni dei dati possono essere eseguite in linea.

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

Dichiarazione a singola variabile

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

Dichiarazione a più variabili

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

Dichiarazione dei dati in linea nell'istruzione SELECT

Quando si utilizza una dichiarazione di dati in linea all'interno di un blocco `SELECT...ENDSELECT` o `SELECT SINGLE` , il carattere `@` deve essere utilizzato come carattere di escape per l'espressione `DATA(lv_cityto)` . Una volta che il carattere di escape è stato utilizzato, anche tutte le altre variabili host devono essere sottoposte a escape (come nel caso di `lv_carrid` seguito).

```
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.
```

Uscite `BERLIN` .

Opzioni di dichiarazione variabile

Diversi tipi di variabili possono essere dichiarati con opzioni speciali.

```
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
```

Leggi Dichiarazione dei dati online: https://riptutorial.com/it/abap/topic/1646/dichiarazione-dei-dati

Capitolo 8: Espressioni regolari

Examples

Sostituzione

L'istruzione `REPLACE` può funzionare direttamente con le espressioni regolari:

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

La variabile `lv_test` valuterà su `The quick bXXwn XXx` .

Ricerca

La dichiarazione `FIND` può funzionare direttamente con le espressioni regolari:

```
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
```

Espressioni regolari orientate agli oggetti

Per operazioni di regex più avanzate è meglio usare `CL_ABAP_REGEX` e le sue classi correlate.

```
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
```

Valutazione di espressioni regolari con una funzione di predicato

Le `matches` funzioni del predicato possono essere utilizzate per valutare le stringhe al volo senza utilizzare dichiarazioni di oggetti.

```
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.
```

Ottenere i sottomatch usando le espressioni regolari OO

Utilizzando il metodo `GET_SUBMATCH` della classe `CL_ABAP_MATCHER` , possiamo ottenere i dati nei gruppi / sottogruppi.

Obiettivo: ottenere il token a destra della parola chiave "Tipo".

```
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.
```

La variabile risultante `lv_smatch` contiene il valore `MARA` .

Leggi Espressioni regolari online: https://riptutorial.com/it/abap/topic/5113/espressioni-regolari

Capitolo 9: Loops

Osservazioni

Quando si esegue il looping su tabelle interne, è generalmente preferibile `ASSIGN` a un simbolo campo piuttosto che eseguire il ciclo `INTO` un'area di lavoro. L'assegnazione dei simboli di campo aggiorna semplicemente il loro riferimento per puntare alla riga successiva della tabella interna durante ogni iterazione, mentre usando `INTO` risulta la linea della tabella che viene copiata nell'area di lavoro, che può essere costosa per le tabelle lunghe / larghe.

Examples

Loop tavolo interno

```
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.
```

Ciclo condizionale

Se solo le linee che corrispondono a una determinata condizione dovrebbero essere inserite nel ciclo, aggiungere `WHERE` può essere aggiunto.

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

Mentre Loop

ABAP offre anche il convenzionale `WHILE` -Loop che viene eseguito fino a quando l'espressione data restituisce false. L' `sy-index` sistema del campo verrà aumentato per ogni passo del ciclo.

```
WHILE condition.
* do something
ENDWHILE
```

Fai il ciclo

Senza alcuna aggiunta, `DO` -Loop funziona all'infinito o almeno fino a quando non viene esplicitamente rimosso dall'interno. L' `sy-index` sistema del campo verrà aumentato per ogni passo del ciclo.

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

L'aggiunta di `TIMES` offre un modo molto conveniente per ripetere il codice (l' `amount` rappresenta un valore di tipo `i`).

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

Comandi generali

Per interrompere i loop, è possibile utilizzare il comando `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.
```

Per saltare alla fase successiva del ciclo, è possibile utilizzare il comando `CONTINUE` .

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

L'istruzione `CHECK` è `CONTINUE` con condizioni. Se la condizione risulta **falsa** , verrà eseguito `CONTINUE` . In altre parole: *il ciclo proseguirà con il passo solo se la condizione è vera* .

Questo esempio di `CHECK` ...

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

... è equivalente a ...

```
DO.
```

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

Leggi Loops online: https://riptutorial.com/it/abap/topic/2270/loops

Capitolo 10: Oggetti ABAP

Examples

Dichiarazione di classe

Le classi ABAP possono essere dichiarate globalmente o localmente . Una classe globale può essere utilizzata da qualsiasi oggetto all'interno del repository ABAP. Al contrario, una classe locale può essere utilizzata solo nell'ambito che è stato dichiarato.

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

CLASS lcl_abap_class IMPLEMENTATION.
ENDCLASS.
```

Costruttore, metodi

Implementazione della classe:

```
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.
```

Esempio di chiamata al metodo:

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

Metodo con parametri (importazione, modifica, esportazione)

Implementazione della classe:

```
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.
```

Esempio di chiamata al metodo:

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

Metodo con parametro di ritorno

Implementazione della classe:

```
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.
```

Esempio di chiamata al metodo:

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

Si noti che i parametri dichiarati con `RETURNING` vengono passati solo per valore.

Ereditarietà - definizione

Informazione

L'ereditarietà consente di derivare una nuova classe da una classe esistente. Lo fai usando l'aggiunta di **INHERITING FROM** nel

Sottoclasse **CLASS DEFINIZIONE EREDITANTE DA** superclasse.

dichiarazione. La nuova sottoclasse di classe eredita tutti i componenti della superclasse di classe esistente. La nuova classe è chiamata la sottoclasse della classe da cui deriva. La classe originale è chiamata la superclasse della nuova classe. Una classe può avere più di una sottoclasse diretta, ma può avere solo una superclasse diretta.

Implementazione di classe

```
CLASS lcl_vehicle DEFINITION.
ENDCLASS.

CLASS lcl_vehicle IMPLEMENTATION.
ENDCLASS.

CLASS lcl_car DEFINITION INHERITING FROM lcl_vehicle.
ENDCLASS.

CLASS lcl_car IMPLEMENTATION.
ENDCLASS.
```

Ereditarietà - Metodi e classi astratti e finali

Informazione

Le aggiunte **ABSTRACT** e **FINAL** alle istruzioni **METHODS** e **CLASS** consentono di definire metodi o classi astratti e finali.

Un metodo astratto è definito in una classe astratta e non può essere implementato in quella classe. Invece, è implementato in una sottoclasse della classe. Le classi astratte non possono essere istanziate.

Un metodo finale non può essere ridefinito in una sottoclasse. Le classi finali non possono avere sottoclassi. Concludono un albero ereditario.

Implementazione della classe:

```
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.
```

Esempio di chiamata al metodo:

```
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( ).
```

Leggi Oggetti ABAP online: https://riptutorial.com/it/abap/topic/2244/oggetti-abap

Capitolo 11: Parola chiave Message Classes / MESSAGE

introduzione

L'istruzione `MESSAGE` può essere utilizzata per interrompere il flusso del programma per visualizzare brevi messaggi all'utente. Il contenuto dei messaggi può essere definito nel codice del programma, nei simboli di testo del programma o in una classe di messaggi indipendente definita in `SE91` .

Osservazioni

La lunghezza massima di un messaggio, inclusi i parametri passati usando `&` , è di 72 caratteri.

Examples

Definizione di una classe di messaggio

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

Il messaggio definito dal sistema può essere memorizzato in una classe di messaggio. Il token `MESSAGE-ID` definisce i `sabapdemos` classe del messaggio per l'intero programma. Se non viene utilizzato, la classe di messaggio deve essere specificata in ogni chiamata `MESSAGE` .

MESSAGGIO con simbolo di testo predefinito

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

Un messaggio visualizzerà il testo memorizzato nel simbolo di testo `i00` all'utente. Poiché il tipo di messaggio è i (come si vede in `i000`), dopo che l'utente ha chiuso la finestra di dialogo, il flusso del programma continuerà dal punto della chiamata `MESSAGE` .

Sebbene il testo non provenga dalla classe di messaggi `za` , è necessario specificare un `MESSAGE-ID` .

Messaggio senza classe di messaggi predefinita

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

Potrebbe essere scomodo definire una classe di messaggio per l'intero programma, quindi è

possibile definire la classe di messaggio che il messaggio proviene `MESSAGE` stessa. Questo esempio mostrerà il messaggio `050` dalla classe messaggio `sabapdemos` .

Messaggistica dinamica

```
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.
```

La chiamata `MESSAGE` sopra è anche chiamata `MESSAGE i050(sapdemos). .`

Passare i parametri ai messaggi

Il simbolo `&` può essere utilizzato in un messaggio per consentire il passaggio dei parametri.

Parametri ordinati

Message `777` of class `sabapdemos` :

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

La chiamata di questo messaggio con tre parametri restituirà un messaggio utilizzando i parametri:

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

Questo messaggio verrà visualizzato come `Message with type E 010 in event START-OF-SELECTION .` Il numero accanto al simbolo `&` indica l'ordine in cui vengono visualizzati i parametri.

Parametri non ordinati

Message `888` of class `sabapdemos` :

```
& & & &
```

La chiamata di questo messaggio è simile:

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

Questo emetterà `param1 param2 param3 param4 .`

Leggi Parola chiave Message Classes / MESSAGE online:
https://riptutorial.com/it/abap/topic/10691/parola-chiave-message-classes---message

Capitolo 12: Programmazione dinamica

Examples

Field-Simboli

I simboli di campo sono equivalenti ai puntatori di ABAP, ad eccezione del fatto che i simboli di campo sono sempre dereferenziati (non è possibile modificare l'indirizzo reale in memoria).

Dichiarazione

Per dichiarare un campo-simbolo è necessario utilizzare la parola chiave `FIELD-SYMBOLS` . I tipi possono essere generici (`ANY [... TABLE]`) per gestire un'ampia varietà di variabili.

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

Assegnazione

I simboli di campo `unassigned` sono `unassigned` alla dichiarazione, il che significa che non indicano nulla. L'accesso a un campo-simbolo non assegnato causerà un'eccezione e, se non eseguito, a un breve dump. Pertanto, lo stato dovrebbe essere controllato con `IS ASSIGNED` :

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

Poiché sono solo riferimenti, non è possibile memorizzare dati reali all'interno. Quindi, dichiarato `DATA` è necessario in ogni caso di utilizzo.

```
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
```

Annullamento dell'assegnazione

A volte potrebbe essere utile resettare un Field-Symbol. Questo può essere fatto usando `UNASSIGN` .

```
UNASSIGN <fs>.
```

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

Utilizzare per tabelle interne

I simboli di campo possono essere usati per modificare tabelle interne.

```
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.
```

Attenzione! I simboli di campo rimangono assegnati anche dopo aver lasciato il ciclo. Se vuoi riutilizzarli in sicurezza, annullali immediatamente.

Riferimenti di dati

Essenziale per i riferimenti dati è l'aggiunta `REF TO` dopo il `TYPE` .

Creazione dinamica di strutture

Se il tipo di una struttura deve essere deciso in runtime, possiamo definire la nostra struttura di destinazione come riferimento ai `data` tipo generico.

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

Per dare `wa` un tipo che usiamo la dichiarazione `CREATE DATA` . Il `TYPE` aggiunta può essere specificato da:

Riferimento:

```
CREATE DATA wa TYPE kna1
```

- *I controlli statici sono attivi quindi non è possibile creare un tipo sconosciuto*

Nome:

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

- *Le parentesi sono necessarie e `lw_name_as_string` contiene il nome dei tipi come stringa.*
- *Se il tipo non è stato trovato, verrà generata un'eccezione di tipo `CX_SY_CREATE_DATA_ERROR`*

Per l'istanziazione di tipi creati dinamicamente è possibile specificare l'aggiunta di `HANDLE` . `HANDLE` riceve un oggetto che eredita da `CL_ABAP_DATADESCR` .

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

- `obj` possono essere creati utilizzando la **R** UN **T** empo **T** ipo **S** ervizi
- perché `dref` è ancora un datareference, deve essere dereferenziato (`->*`) per essere usato come datacontainer (normalmente fatto tramite Field-Symbols)

Servizi di tipo RunTime

RunTime Type Services (*abbreviazione:* **RTTS**) sono utilizzati per:

- creazione di tipi (Creazione di tipo RunTime; *corto:* **RTTC**)
- tipi di analisi (RunTime Type Identification; *short:* **RTTI**)

Classi

```
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` è la classe base. Implementa i metodi necessari per descrivere:

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

Leggi Programmazione dinamica online:
https://riptutorial.com/it/abap/topic/4442/programmazione-dinamica

Capitolo 13: Programmi modello

Sintassi

- CLASS DEFINITION ABSTRACT FINAL rende la classe del programma essenzialmente statica in quanto i metodi di istanza non potrebbero mai essere utilizzati. L'intenzione è di mantenere la classe minima.

Examples

Programma OO con metodi di eventi essenziali

```
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( ).
```

Leggi Programmi modello online: https://riptutorial.com/it/abap/topic/10552/programmi-modello

Capitolo 14: stringhe

Examples

letterali

ABAP offre tre diversi operatori per dichiarare variabili tipo stringa o char

simboli	Tipo interno	Lunghezza	Nome
'...'	C	1-255 caratteri	letterali del campo di testo
`...`	CString	0-255 caratteri	letterali stringa di testo
| ... |	CString	0-255 caratteri	letterali modello

Si noti che l'intervallo di lunghezza si applica solo ai valori codificati. Le variabili di `CString` internamente hanno una lunghezza arbitraria mentre le variabili di tipo `C` hanno sempre una lunghezza fissa.

Modelli di stringa

I modelli di stringhe sono un modo conveniente per mescolare stringhe letterali con valori di variabili:

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

Può anche formattare cose come le date. Per utilizzare il formato data dell'utente connesso:

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

Sono supportate le chiamate e le espressioni del metodo funzionale:

```
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' )
}|.
```

> **Attenzione!** L'implementazione diretta di risultati temporanei (come le chiamate al metodo) all'interno di modelli di stringhe può portare a enormi problemi di prestazioni (maggiori informazioni qui). Mentre lo si utilizza all'interno di istruzioni eseguite raramente, è possibile che il programma rallenti rapidamente nei cicli.

Concatenazione di stringhe

Le variabili stringa e char possono essere concatenate utilizzando il comando ABAP `CONCATENATE` .

È necessaria una variabile extra per la memorizzazione dei risultati.

Esempio:

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

Abbreviazione

Le versioni più recenti di ABAP offrono una variante molto breve di concatenazione usando && (operatore Chaining).

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

> **Attenzione!** Vale la pena notare che l'utilizzo di risultati temporanei in combinazione con l'operatore Chaining all'interno di loop può portare a enormi problemi di prestazioni a causa delle crescenti istruzioni di copia (leggete di più qui).

Leggi stringhe online: https://riptutorial.com/it/abap/topic/3531/stringhe

Capitolo 15: Tabelle interne

Examples

Tipi di tabelle interne

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

Tabella standard

Questa tabella contiene tutte le voci memorizzate in modo lineare e i record sono accessibili in modo lineare. Per le tabelle di grandi dimensioni, l'accesso alla tabella può essere lento.

Tabella ordinata

Richiede l'aggiunta `WITH UNIQUE | NON-UNIQUE KEY` . La ricerca è veloce a causa dell'esecuzione di una ricerca binaria. Le voci non possono essere aggiunte a questa tabella in quanto potrebbero rompere l'ordinamento, pertanto vengono sempre inserite utilizzando la parola chiave `INSERT` .

Tabella tritata

Richiede l'aggiunta `WITH UNIQUE | NON-UNIQUE KEY` . Utilizza un algoritmo di hashing proprietario per mantenere le coppie chiave-valore. Le ricerche teoriche possono essere lente come la tabella `STANDARD` ma in pratica sono più veloci di una tabella `SORTED` richiede una quantità di tempo costante indipendentemente dalle dimensioni della tabella.

Dichiarazione delle tabelle interne ABAP

Dichiarazione della tabella interna basata sulla definizione del tipo locale

```
" 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.
```

Dichiarazione basata sulla tabella del database

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

Dichiarazione della tabella interna in linea

Richiede la versione ABAP> 7.4

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

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

Tabella interna con dichiarazione delle righe di intestazione

In ABAP ci sono tabelle con linee di intestazione e tabelle senza linee di intestazione. Le tabelle con le linee di intestazione sono un concetto precedente e non dovrebbero essere utilizzate in un nuovo sviluppo.

Tabella interna: tabella standard con / senza riga di intestazione

Questo codice dichiara la tabella `i_compc_all` con la struttura esistente di `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.
```

Tabella interna: tabella con hash con / senza riga di intestazione

```
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
```

Dichiarazione di un'area di lavoro per tabelle senza intestazione

Un'area di lavoro (comunemente abbreviata *wa*) ha la stessa struttura della tabella, ma può contenere solo una riga (una WA è una struttura di una tabella con una sola dimensione).

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

Leggere, scrivere e inserire nelle tabelle interne

Leggere, scrivere e inserire nelle tabelle interne con una riga di intestazione:

```
" 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.
```

> Promemoria: le tabelle interne con le linee di intestazione sono vietate in contesti orientati agli oggetti. L'utilizzo di tabelle interne *senza* righe di intestazione è sempre consigliato.

Leggere, scrivere e inserire nelle tabelle interne senza una riga di intestazione:

```
" 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
```

Leggi Tabelle interne online: https://riptutorial.com/it/abap/topic/1647/tabelle-interne

Capitolo 16: Test unitario

Examples

Struttura di una classe di test

Le classi di test vengono create come classi locali in un controllo di unità speciale incluso.

Questa è la struttura di base di una classe di test:

```
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.
```

Qualsiasi metodo dichiarato con `FOR TESTING` sarà un test unitario. `setup` è un metodo speciale che viene eseguito prima di ogni test.

Separare l'accesso ai dati dalla logica

Un principio importante per il test delle unità è quello di separare l'accesso ai dati dalla logica aziendale. Una tecnica efficiente per questo è definire le interfacce per l'accesso ai dati. La tua classe principale usa sempre un riferimento a quell'interfaccia invece di leggere direttamente o scrivere dati.

nel codice di produzione alla classe principale verrà assegnato un oggetto che avvolge l'effettivo accesso ai dati. Questa potrebbe essere una dichiarazione selezionata, chiamate di mudule di funzione, qualsiasi cosa veramente. La parte importante è che questa classe non dovrebbe eseguire nient'altro. Nessuna logica

Durante il test della classe principale, gli dai un oggetto che serve invece dati statici e falsi.

Un esempio per accedere alla tabella `SCARR`

Interfaccia di accesso ai dati `ZIF_DB_SCARR`:

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

Classe dati falsi e classe di test:

```
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.
```

L'idea qui è che nel codice di produzione, `ZCL_MAIN_CLASS` otterrà un oggetto `ZIF_DB_SCARR` che esegue un `SELECT` e restituisce l'intera tabella mentre il test dell'unità è eseguito su un set di dati statico definito proprio lì nell'unità di test.

Leggi Test unitario online: https://riptutorial.com/it/abap/topic/3999/test-unitario

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con ABAP	Christian, Community, gkubed, Jagger, mkysoft
2	ABAP GRID List Viewer (ALV)	Achuth hadnoor, gkubed
3	Apri SQL	AKHIL RAJ, gkubed
4	Commenti	4444, Christian, gkubed
5	Controllo delle dichiarazioni di flusso	Community, gkubed, maillard
6	Convenzioni di denominazione	mkysoft
7	Dichiarazione dei dati	Christian, gkubed
8	Espressioni regolari	AKHIL RAJ, gkubed, maillard
9	Loops	Christian, Community, gkubed, Stu G
10	Oggetti ABAP	Community, Michał Majer, Thomas Matecki
11	Parola chiave Message Classes / MESSAGE	gkubed
12	Programmazione dinamica	Community, gkubed
13	Programmi modello	nath
14	stringhe	Achuth hadnoor, Community, maillard, nexxus, Suncatcher
15	Tabelle interne	Community, gkubed, Michał Majer, Rahul Kadukar, Thorsten Niehues
16	Test unitario	maillard