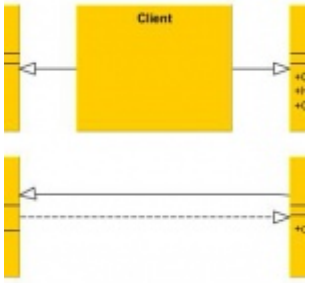
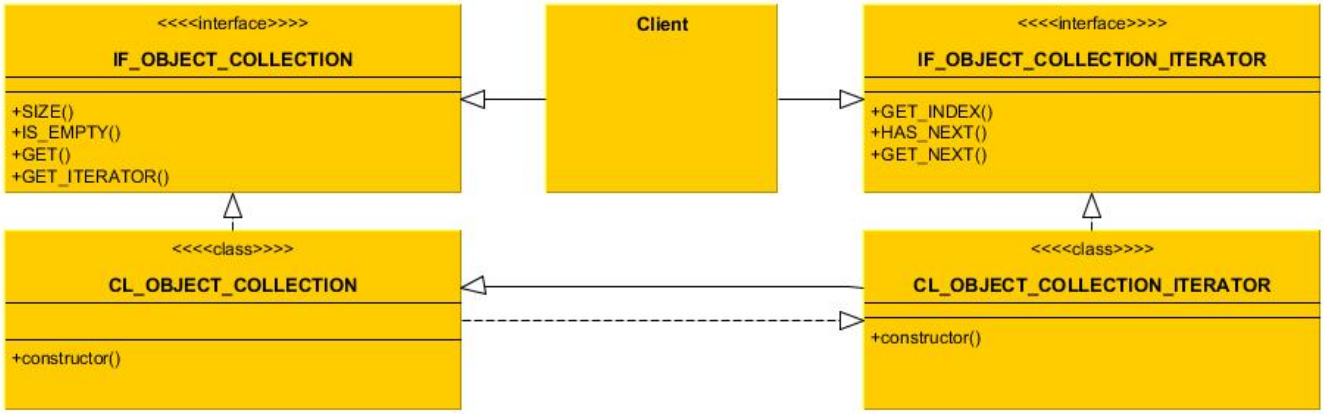


Iterator [Design Pattern]

Es bleibt mir eigentlich nur das Abschreiben von ABAP-Guru Naimesh: Iterator Pattern

Ich kann kein UML, deswegen verweise ich auf die exzellent aufbereitete Seite von Naimesh.

Aber damit ich auch was dazu lerne, habe ich mit yEd das Diagramm einfach mal abgezeichnet:



Was macht der Iterator?

Der Iterator – Wiederholer – ist ein Entwurfsmuster, das dazu verwendet werden kann, mehrere identische Objekte zu verwalten. Der Zugang zu den einzelnen Objekten erfolgt inder Regel über die Methode GET_NEXT in einer Schleife um alle Elemente abzuarbeiten. Der Zugriff kann ebenfalls über eine Index erfolgen. Ob ein Iterator noch zu bearbeitende Elemente hat, wird über die Methode HAS_NEXT abgefragt.

Verwendungsmöglichkeiten

Das Iterator-Pattern lässt sich immer dann gut verwenden, wenn ein Objekte mehrere Unterobjekte enthält, deren Anzahl unterschiedlich sein kann. Das Open-Source-Projekt ABAP2XLS verwendet den Iterator beispielsweise für die Verwaltung von Sheets, Styles oder Ranges.

Der Iterator lässt sich also immer dann gut benutzen, wenn ein Objekt eine unterschiedliche Anzahl von Objekten hat. Neue Objekte werden dann mittels *add(object)* zu einer *Collection* hinzugefügt.

Unterstützung durch SAP

SAP unterstützt die Verwendung des Iterator Pattern durch die Klassen CL_OBJECT_COLLECTION_ITERATOR und CL_OBJECT_COLLECTION. Die beiden Klassen werden in dem Beispielprogramm verwendet.

Im Gegensatz zu Naimesh denke ich jedoch nicht, dass die von SAP zur Verfügung gestellten Klassen nutzlos sind. Sie lassen sich sehr gut verwenden, wie das unten gezeigte Beispiel beweist.

Beispiel

In dem folgenden Beispiel habe ich ein Objekt für einen Buchungskreis definiert (LCL_BUKRS) und alle Buchungskreise aus der Tabelle T001 eingelesen und zu einer Collection hinzugefügt. Die Abarbeitung erfolgt dann über eine Schleife, die Abfragt, ob noch Elemente vorhanden sind.

Der Iterator besitzt keine Methode um den Index wieder auf „1“ zu setzen. Um das zu erreichen, muss ein aktuelles ITERATOR-Objekt geholt werden.

Weitere Informationen

Ein Beispiel im SAP-Standard ist das Programm TCL_ITERATOR_EXAMPLE

Das Beispiel benutzt eigene Klassen für die Iteratoren.

```
*&---------------------------------------------------------------------*
*& This program is a short introduction in using the ABAP Foundation
*& classes CL_TCL_AGGREGATE and CL_TCL_ITERATOR.
*& The development of this classes was inspired by the book
*& "Design Patterns, Elements of Reusable Object-Oriented Software".
*& This example demonstrates the power of the Iterator design pattern
*& (and ABAP Objects too :-). Using the Iterator design pattern you are
*& able to design and develop robust programs with ability of easy
*& functionalty improvements. High performance programms with the main
*& focus to save every possible microsecond should use other concepts,
*& if the OOO (Object Orientation Overhead ;-) is an issue.
*&---------------------------------------------------------------------*
```

Code

```
REPORT zz_dp_iterator.

CLASS lcl_bukrs DEFINITION.

 PUBLIC SECTION.
 DATA ms_t001 TYPE t001.
 METHODS constructor IMPORTING bukrs TYPE bukrs.
 METHODS get_info RETURNING VALUE(t001) TYPE t001.

ENDCLASS.

CLASS lcl_bukrs IMPLEMENTATION.
```

```
 METHOD constructor.
 SELECT SINGLE * FROM t001 INTO ms_t001 WHERE bukrs = bukrs.
 ENDMETHOD.
 METHOD get_info.
 t001 = ms_t001.
 ENDMETHOD.
ENDCLASS.

CLASS lcl_main DEFINITION.
 PUBLIC SECTION.
 METHODS start.
 METHODS get_iterator RETURNING VALUE(itERATOR) TYPE REF TO
CL_OBJECT_COLLECTION_ITERATOR.
 METHODS GET_OBJECT IMPORTING INDEX TYPE I
 RETURNING VALUE(OBJECT) TYPE REF TO OBJECT .

 PROTECTED SECTION.
 DATA MR_OBJECT_COLLECTION TYPE REF TO CL_OBJECT_COLLECTION.
ENDCLASS.

CLASS lcl_main IMPLEMENTATION.

 METHOD start.
 DATA lr_bukrs TYPE REF TO lcl_bukrs.
 DATA lt_bukrs TYPE STANDARD TABLE OF bukrs.
 DATA lv_bukrs TYPE bukrs.
 SELECT bukrs FROM t001 INTO TABLE lt_bukrs.
 CREATE OBJECT mr_object_collection.

 LOOP AT lt_bukrs INTO lv_bukrs.
 CREATE OBJECT lr_bukrs EXPORTING bukrs = lv_bukrs.
 mr_object_collection->add( lr_bukrs ).

 ENDLOOP.
 ENDMETHOD.

 METHOD get_iterator.
 iterator = mr_object_collection->if_object_collection~get_iterator( ).
 ENDMETHOD.

 METHOD get_object.
 object = mr_object_collection->get( index ).
 ENDMETHOD.

ENDCLASS.

DATA gr_main TYPE REF TO lcl_main.

START-OF-SELECTION.
 CREATE OBJECT gr_main.
```

```
DATA gr_iterator TYPE REF TO cl_object_collection_iterator.
DATA gr_bukrs TYPE REF TO lcl_bukrs.
DATA gv_butxt TYPE string.
DATA gv_index TYPE i.
DATA gv_bukrs TYPE bukrs.

gr_main->start( ).
gr_iterator = gr_main->get_iterator( ).
WHILE gr_iterator->has_next( ).
gr_bukrs ?= gr_iterator->get_next( ).
gv_butxt = gr_bukrs->get_info( )-butxt.
WRITE: / gv_butxt.
IF gv_butxt = 'SAP A.G.'.
gv_index = gr_iterator->get_index( ).
ENDIF.
ENDWHILE.


IF gv_index IS NOT INITIAL.
gr_bukrs ?= gr_main->get_object( gv_index ).
gv_bukrs = gr_bukrs->get_info( )-bukrs.
WRITE: / gv_bukrs COLOR COL_GROUP.
ENDIF.

gr_iterator = gr_main->get_iterator( ).

WHILE gr_iterator->has_next( ).
gr_bukrs ?= gr_iterator->get_next( ).
gv_butxt = gr_bukrs->get_info( )-butxt.
WRITE: / gv_butxt.
IF gv_butxt = 'SAP A.G.'.
gv_index = gr_iterator->get_index( ).
ENDIF.
ENDWHILE.
```