Automated Unit Testing with ABAP

Requirements Document for Associated Exercise Programs

This document describes the requirements for the exercise programs associated with the book Automated Unit Testing with ABAP.

Table of Contents

1 Overview of exercise programs

Completed sample exercise programs are provided to accompany many of the chapters in the book Automated Unit Testing with ABAP. These are to be created, edited and run in an SAP environment supporting ABAP programming development. The student is expected to copy the first sample exercise program to a student-specific copy and to use this as a starting point for subsequent exercises. Each exercise program is slightly different from its predecessor and covers a single change or set of changes to be applied relevant to the associated chapter of the book. Refer to the section **1.2 Exercise program naming convention** for more information on program naming conventions suggested for use with new student copies of exercise programs.

The sample exercise programs all exhibit the following characteristics:

1. The title of the program is the same as the name of the program.

2. The program property associated with Unicode compliance is set “on”. Students are advised to retain this setting in their own exercise programs.

3. Each sample exercise program has a comment block at the top of the code indicating how to define selection texts associated with the program. This can be used to determine whether all the components of the program are defined correctly.

1.1 Determining how to apply changes to exercise programs

In addition to the explanations outlined in this document, the following additional aid is recommended for determining the differences between two adjacent versions of exercise programs:

- Perform a comparison of the differences between two adjacent versions by following these simple steps in the SAP environment in which the exercises are being performed:
  1. Invoke transaction SE39.
  2. From menu, select: Utilities > Settings.
  3. On tab ABAP Editor, sub-tab Splitscreen, select check-mark to indicate "Ignore Indentations" and press enter.
  4. Specify the name of the object representing the preceding version in the Left Program slot, the newer version in the Right Program slot, and press Display (or simply press Enter).
  5. Press the "Comparison On" button appearing on the button bar (or press CTRL+F4).
  6. Alternate pressing the "Next Difference from Cursor" and "Next Identical Section from Cursor" (easiest to do this by holding CTRL+SHIFT and alternately pressing F9 and F11). Both the left and right sections are scrolled forward together as the next equal or different line is located.

1.2 Exercise program naming convention

The exercise programs all are named using the following naming convention:

Character position	Value	Notes
1	Z	First character of all user programs must begin with Y or Z.
2 – 4	AUT	Acronym for ABAP Unit Testing.
5 – 7	Topic identification number	A 3-digit number corresponding to a topic presented in the book Automated Unit Testing with ABAP. The numbers represent the following topics: ABAP Unit Testing 101 – Creating Your First Unit Test

		ABAP Unit Testing 102 – Writing Additional Unit Tests ABAP Unit Testing 103 – Writing Unit Tests for Function Modules ABAP Unit Testing 104 – Writing Unit Tests for Global Classes ABAP Unit Testing 105 – How Certain ABAP Statements Affect Unit Testing ABAP Unit Testing 106 – How Unit Testing Enables Confident Refactoring ABAP Unit Testing 107 – Diagnosing the Absence of Sufficient Test Data ABAP Unit Testing 108 – Creating and Using Fabricated Test Data ABAP Unit Testing 109 – Gaining Control Over References to Modifiable Global Variables Within Subroutines ABAP Unit Testing 201 – Gaining Control Over Unit Test Coverage of Input ABAP Unit Testing 202 – Gaining Control Over Unit Test Coverage of Output ABAP Unit Testing 301 – Introducing a Test Double for Input ABAP Unit Testing 302 – Introducing a Test Double for Output ABAP Unit Testing 401 – Introducing a Service Locator ABAP Unit Testing 402 – Introducing a Service Factory ABAP Unit Testing 501 – Gaining Control Over Global Class Dependencies ABAP Unit Testing 502 – Gaining Control Over Function Module Dependencies ABAP Unit Testing 503 – Gaining Control Over Message Statements ABAP Unit Testing 504 – Gaining Control Over List Processing Statements ABAP Unit Testing 601 – Detecting Missing Service Locators ABAP Unit Testing 701 – Using the ABAP Test Double Framework
8	Alphabetic character	Unique appendage distinguishing program names related to the same topic identification number, starting with A and proceeding through the alphabet as required.

The naming convention suggested for use by the student is modeled after the naming convention shown above, with the 3-character initials of the student to be placed between the first character “Z” and the “AUT” characters otherwise occupying the next three character positions, e.g.: ZLVBAUT101B for Ludwig Von Beethoven. All new exercise programs should be assigned package $TMP.

1.3 Exercise program intent

The exercise programs are intended to assist the student in understanding the concepts associated with ABAP Unit Testing. They are written specifically so that the same functionality provided by the first exercise program continues throughout the remaining exercises, with the expectation that the student will become familiar with the basic functionality with the first exercise program, and thereafter no longer will need to learn any new functionality other than what is being introduced with the new applicable concepts. Accordingly, these exercise programs focus on learning concepts and do not serve as suitable models for actual user programs, with the following among some of the reasons:

- To facilitate simplicity, these exercise programs are devoid of security checking and robust exception checking, aspects of programming normally present in production programs.

- To facilitate easy comparisons between two adjacent exercise programs via transaction SE39, all code associated with one exercise program is contained within a single object.

1.4 Loading the accompanying exercise programs into your ABAP repository

Since you are reading this document then probably you already have downloaded from the internet the corresponding zip file containing this exercises workbook and the text files for its accompanying 181 executable ABAP exercise programs. Included among these is the text file of an ABAP program utility to automatically load all of the exercise programs into the

ABAP repository of the SAP training environment where you will be performing the exercises.

The file name of this utility is **zautupl.txt**. Manually create a new program in your ABAP repository using the ABAP source code from this file, save and activate it. Execute this program and specify the directory where the exercise program source text files have been downloaded. Unless the standards for you site require some other prefix designation, retain the specified prefix 'ZAUT' for the names of the new programs to be created and press Execute. An ALV report is presented describing all the activity this utility would otherwise perform if it were to be executed in "update" mode. Review the report. If there are no errors, then run the program again, this time placing a check mark in the "Update mode?" checkbox. Afterward, all associated executable ABAP exercise programs will have been created in the ABAP repository of the SAP training environment .

> **Note:** All of these executable ABAP exercise programs should have been marked as "activated" by the load utility despite some of them containing syntax errors due to components that have not yet been created in the ABAP repository. These missing components will be created accordingly as part of the exercises described by this document.

1.5 Insuring test data records exist

The basic functionality underlying each of the exercise programs is to present a simple ALV report of records selected from table SFLIGHT, specifically, records having a carrier id value (CARRID) of "LH" (Lufthansa), "UA" (United Airlines) and "AA" (American Airlines).

Execute program BALVEX02 or use transaction SE16 (or similar) to determine whether table SLFIGHT contains any records having these carrier id values. If none are found, then use test data generator program SAPBC_DATA_GENERATOR to generate bulk records for these carrier ids or use transaction BC_GLOBAL_SFLGH_CREA (utility program SAPBC_GLOBAL_SFLIGHT_CREATE) to generate specific records individually.

2 Organization of requirements

This requirements document provides each exercise program with an explanation of what the student should do to change the program and what should occur as a result of both executing the program and executing the corresponding unit tests. In those cases where the student reaches a point where it becomes difficult to decide how to apply a new change, it is recommended to run the source code comparator between the previous and current versions of the two relevant sample exercise programs to see how changes were applied. A recommended process to follow for running the comparator is described in **Section 1.1**.

Sample exercise program ZAUT101A is the starting point. It is written entirely using classic ABAP statements. Indeed, the author of the program committed some breaches of what now are considered best ABAP programming practices, such as using the obsolete FORM-ENDFORM subroutine construct, defining global variables, using references to screen variables throughout the subroutines and defining some subroutines with no signature, relying instead on passing values via global variables. This was done intentionally to provide a realistic example of the style of programming the student is likely to encounter in any one of thousands of standard SAP or customized ABAP programs[1]. The program is written intentionally using old and outdated features of ABAP programming in order to provide a convincing case that even old legacy code is capable of having effective automated unit tests created for it.

By copying this program to your own student copy and applying changes described by these requirements, and doing this repeatedly for each new requirement, this simple program eventually is transformed into a program having most of its code covered by automated unit tests.

Note: This document was prepared using the application **Writer** of the **LibreOffice** suite of applications to create an Open Text Document (file extension .odt), then generated into a corresponding Portable Document Format document (file extension .pdf). The .pdf format of this document was created under the assumption that most students will be able to read a .pdf document without the need to install any new software to do so.

When it comes to applying the recommended ABAP source code changes noted in this document to the next version of the exercise program, the student may find themselves opting to copy and paste those changes provided with each new requirement. It was found that using this technique when copying source code lines from a .pdf document, line indenture and consecutive spaces formatting are not observed. Accordingly, this document has been provided with a supplemental document in .txt format to provide source code lines which, when copied and pasted into an ABAP program, will retain the corresponding indenture and consecutive spaces. The file name is:

o Automated Unit Testing with ABAP exercises workbook supplemental source code.txt

Perform a search in this document using the corresponding exercise program name (ZAUTnnnx) to find the location of the associated source code lines to be copied and pasted.

Though not used with the sample exercise programs, the student may opt to break the program into a series of components connected through INCLUDE statements.

1 Refer to the book Official ABAP Programming Guidelines (Horst Keller, Wolf Hagen Thümmel; Galileo Press, 2010) for more information on avoiding such pitfalls.

3 ABAP Unit Testing 101 – Creating Your First Unit Test

This section describes the requirements for the exercise programs associated with the Chapter 6 section titled Introducing a Simple Unit Test in the book Automated Unit Testing with ABAP.

3.1 Exercise 1

Program: ZAUT101A

Program overview

This program has a selection screen on which the user specifies the following information:

- Airline (required)
- Airfare discount percentage (optional)
- Whether to display using ALV classic list or ALV grid

After providing values in the fields on the selection screen, the user presses Execute and an ALV report is produced containing rows from the SFLIGHT table conforming to the specified airline.

This is the starting program model. It is written using only the classic procedural ABAP capabilities – that is, it is devoid of containing or using any customized local or global classes or any standard SAP global classes. Its intent is merely to produce a report using the information provided on the selection screen.

Become familiar with the code in this program. You will be using it as the starting model for applying changes corresponding to all of the concepts covered in the book Automated Unit Testing with ABAP.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Error message "No flights match carrier AA" appears at bottom of screen.

Test

Action: While this program is displayed in the ABAP editor (SE38), select from menu:
Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Warning message appears at bottom of screen indicating program has no unit tests defined.

Remarks

This illustrates the result of attempting to invoke the test runner of the Automated Unit Test Framework against a program that has no unit tests defined for it. You should get the same result when attempting this with any of the hundreds of other customized ABAP components at your site which have never had automated unit tests written for them.

Let’s take a moment to observe and record some of the issues we see associated with this program. As we continue through the exercises we will record the version of the program where the issue is eventually resolved. For now, we should notice the following issues pending resolution:

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A		No test for code in subroutine adjust_flight_revenue
3	ZAUT101A		No test for code in subroutine apply_flight_discount
4	ZAUT101A		No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A		No test for code in subroutine get_flights_via_carrier
6	ZAUT101A		No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A		No test for code in subroutine set_alv_field_catalog
9	ZAUT101A		No test for code in subroutine set_alv_function_module_name
10	ZAUT101A		No test for code in subroutine show_flights
11	ZAUT101A		No test for code in subroutine show_flights_count
12	ZAUT101A		No example of ABAP Unit test for function module
13	ZAUT101A		No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant

3.2 Exercise 2

Program: ZAUT101B

Requirements

Reason for change

- Begin the process of providing a unit test for subroutine set_alv_field_catalog.

Changes to be applied

1. Copy the following lines to the end of the program (be certain to use file Automated Unit Testing with ABAP exercises workbook supplemental source code.txt to obtain these lines of code in a format that will retain formatting; Refer to section **2 - Organization of requirements** in this document for more information about the purpose of this file):

```
*=====================================================================
*
*   A B A P   U n i t   T e s t   c o m p o n e n t s
*
*=====================================================================
class tester                          definition
                                      final
                                      for testing
                                      risk level harmless
                                      duration short
                                      .
  private section.
    methods      : set_alv_field_catalog
                   .
endclass.
class tester                          implementation.
  method set_alv_field_catalog.
  endmethod.
endclass.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Error message "No flights match carrier AA" appears at bottom of screen.

Test

Action: While this program is displayed in the ABAP editor (SE38), select from menu:
Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 0 test classes, 0 test methods.

Remarks

With this version we have included a new unit test class called tester. It is a unit test class because it has the “for testing” clause on its class definition statement. It has a single private method called set_alv_field_catalog which has an empty implementation.

This time the test runner of the Automated Unit Test Framework recognized that there was a test class associated with the program, hence the difference between the warning message resulting from the previous version and the status message with this version. Despite an associated test class being present, it has no methods marked as “for testing”, so in the message produced by the test runner we see “0 test methods”.

As with most automated testing frameworks, the ABAP Automated Unit Test Framework identifies the automated unit tests defined for a component through a process known as “Test Discovery”, a unit testing pattern cataloged by Gerard Meszaros (see xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 393). Here it has identified that this program has a class marked as “for testing”.

3.3 Exercise 3

Program: ZAUT101C

Requirements

Reason for change

- Continue the process of providing a unit test for subroutine set_alv_field_catalog.

Changes to be applied

1. Apply "for testing" to test method set_alv_field_catalog of class tester:

```
methods      : set_alv_field_catalog
                 for testing
               .
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Error message "No flights match carrier AA" appears at bottom of screen.

Test

Action: While this program is displayed in the ABAP editor (SE38), select from menu:
Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 1 test methods.

Remarks

With this version we have indicated that method set_alv_field_catalog of class tester is now a “test method”. Now when the unit test is run the test runner produces a message indicating “1 test method”. Also notice that test method set_alv_field_catalog still has an empty implementation.

3.4 Exercise 4

Program: ZAUT101D

Requirements

Reason for change

- Complete the process of providing a unit test for subroutine set_alv_field_catalog.

Changes to be applied

1. Include in test method set_alv_field_catalog code to test call to subroutine set_alv_field_catalog. The method should look like the following when completed:

```
method set_alv_field_catalog.
  data         : alv_fieldcat_stack
                          type slis_t_fieldcat_alv
```

```
              .
  " Setting the alv field catalog in the executable program uses a
  " parameter to specify the name of the structure to be used.  If
  " this name is invalid, no field catalog entries will result.  Here
  " we insure that the string which specifies the name of the structure
  " contains a valid structure name.
  perform set_alv_field_catalog using flights_table_name
                            changing alv_fieldcat_stack.
  call method cl_abap_unit_assert=>assert_not_initial
    exporting
      act                      = alv_fieldcat_stack
      msg                      = 'ALV fieldcatalog is empty'
      .
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Error message "No flights match carrier AA" appears at bottom of screen.

Test

Action: While this program is displayed in the ABAP editor (SE38), select from menu:
Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test method set_alv_field_catalog triggers failure; Status message indicates Processed: 1 program, 1 test classes, 1 test methods.

Remarks

With this version we have provided test method set_alv_field_catalog with an activity to perform and an assertion to check – specifically, call subroutine set_alv_field_catalog and assert that global table variable alv_fieldcat_stack contains records . When this unit test is executed its assertion fails – in this case, the global table variable alv_fieldcat_stack remains empty after the call to subroutine set_alv_field_catalog. Since the assertion is expecting this table to be populated with rows, the assertion fails and we are presented with the ABAP Unit: Results Display report highlighting the failure.

Let’s register in our issues list that this version resolves issue #8.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A		No test for code in subroutine adjust_flight_revenue
3	ZAUT101A		No test for code in subroutine apply_flight_discount
4	ZAUT101A		No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A		No test for code in subroutine get_flights_via_carrier
6	ZAUT101A		No test for code in subroutine get_flight_revenue

#	Identified	Resolved	Description
7	ZAUT101A		No test for code in subroutine present_report
8	**ZAUT101A**	**ZAUT101D**	**No test for code in subroutine set_alv_field_catalog**
9	ZAUT101A		No test for code in subroutine set_alv_function_module_name
10	ZAUT101A		No test for code in subroutine show_flights
11	ZAUT101A		No test for code in subroutine show_flights_count
12	ZAUT101A		No example of ABAP Unit test for function module
13	ZAUT101A		No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant

3.5 Exercise 5

Program: ZAUT101E

Requirements

Reason for change

- Fix the production code bug identified by the previous version unit test.

Changes to be applied

1. Change value for global constant flights_table_name from XFLIGHT to SFLIGHT.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV list display, but for wrong airline.

Test

Action: While this program is displayed in the ABAP editor (SE38), select from menu:
Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 1 test methods.

Remarks

With this version we have corrected the table name specified for flights_table_name. When the unit tests are run, the same assertion that previously failed in test method set_alv_field_catalog now passes. Notice that when the unit tests pass we simply see a status message appear indicating statistics about the associated unit tests. There is no report we need to go browse in order to determine that there are no unit test failures

We might have been able to find this bug and correct it ourselves without the use of an automated test to find it for us, but this is only the beginning of using automated tests to uncover problems with the production logic, and this unit test is now a permanent part of this program. This unit test will be executed every time the developer requests the unit tests for this program to be run, and we can rely on it to determine whether this table name ever gets corrupted in the future by changes applied to the program.

We also see that running the program no longer results in an error message but presents an ALV report containing a list of flights from the respective transparent table. Unfortunately, the rows presented in the ALV report are not representative of the airline we had specified.

3.6 Exercise 6

Program: ZAUT101F

Requirements

Reason for change

- Use functional method call syntax to make unit test assertion.

Changes to be applied

1. Change test method set_alv_field_catalog of class tester to use the newer syntax for invoking method assert_not_initial of class cl_abap_unit_assert:
   - Replace this statement ...

```
call method cl_abap_unit_assert=>assert_not_initial
  exporting
    act                       = alv_fieldcat_stack
    msg                       = 'ALV fieldcatalog is empty'
    .
```

… with this statement:

```
cl_abap_unit_assert=>assert_not_initial(
  act                         = alv_fieldcat_stack
  msg                         = 'ALV fieldcatalog is empty'
  ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV list display, but for wrong airline.

Test

Action: While this program is displayed in the ABAP editor (SE38), select from menu:
Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 1 test methods.

Remarks

With this version we have simply replaced the statement calling the method of class cl_abap_unit_assert with the newer statement format for calling methods. There are no noticeable differences between this version and the previous version when executing the report or running the tests. From this point forward we will be using this newer method call syntax to invoke the assertion methods of class cl_abap_unit_assert.

4 ABAP Unit Testing 102 – Expanding Unit Test Coverage

This section describes the requirements for the exercise programs associated with the Chapter 6 section titled Expanding Unit Test Coverage in the book Automated Unit Testing with ABAP.

4.1 Exercise 7

Program: ZAUT102A

Requirements

Reason for change

- Implement unit test for subroutine get_flights_via_carrier.

Changes to be applied

1. Add new test method get_flights_via_carrier to class tester to test call to subroutine get_flights_via_carrier:
   - Add method definition for get_flights_via_carrier to the private section of class tester after the definition for method set_alv_field_catalog:

```
  methods      : o
                 o
                 o
               , get_flights_via_carrier
                   for testing
```

   - Include the following method implementation after the implementation for method set_alv_field_catalog:

```
method get_flights_via_carrier.
  constants    : lufthansa      type s_carr_id value 'LH'
               , united_airlines
                                type s_carr_id value 'UA'
               , american_airlines
                                type s_carr_id value 'AA'
               .
  data         : failure_message
                                type string
               , flights_entry  like line
                                  of flights_stack
               , carrier_id_stack
                                type table
                                  of s_carr_id
               , carrier_id_entry
                                like line
                                  of carrier_id_stack
               .
  " This unit test is modelled after the example unit test presented
  " in the book "ABAP Objects - ABAP Programming in SAP NetWeaver",
  " 2nd edition, by Horst Keller and Sascha Kruger (Galileo Press,
  " 2007, ISBN 978-1-59229-079-6).  Refer to the sample listing 13.3
  " starting on page 964.  Here we insure that the list of flights
  " retrieved contains only those flights for the specified carrier.
  append: lufthansa             to carrier_id_stack
        , united_airlines       to carrier_id_stack
        , american_airlines     to carrier_id_stack
        .
  loop at carrier_id_stack
     into carrier_id_entry.
    concatenate 'Selection of'
                carrier_id_entry
                'gives different airlines'
           into failure_message separated by space.
```

```
    perform get_flights_via_carrier using carrier_id_entry.
    " We have specified a quit parameter for the next assertion.
    " The default action is to terminate the test method upon encountering
    " an error.  We do not want to terminate this test method with the
    " first error because we intend to run this test for multiple carriers
    " as identified in the outer loop, allowing ABAP Unit test errors to
    " be issued for whichever carriers they apply.
    " Notice also that the vale specified for the quit parameter is a
    " constant defined in class cl_aunit_assert.  Class cl_aunit_assert
    " is the name of the first generation of ABAP Unit assertion class.
    " It still exists and still can be used, but SAP has since superseded
    " this class with the more descriptively named assertion class
    " cl_abap_unit_assert.  We are using the old class name here because its
    " static attributes were not made available to class cl_abap_unit_assert.
    loop at flights_stack
       into flights_entry.
      cl_abap_unit_assert=>assert_equals(
        act                    = flights_entry-carrid
        exp                    = carrier_id_entry
        msg                    = failure_message
        quit                   = cl_aunit_assert=>no
        ).
      if flights_entry-carrid ne carrier_id_entry.
        exit. " loop at flights_stack
      endif.
    endloop.
  endloop.
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV list display, but for wrong airline.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Results Display report indicates test method get_flights_via_carrier triggers two failure messages; Status message indicates Processed: 1 program, 1 test classes, 2 test methods.

Remarks

With this version we have defined a second unit test method for class tester: get_flights_via_carrier. Its implementation consists of 3 calls to subroutine get_flights_via_carrier, with each call specifying a different airline. This new unit test method is modeled after the one found in the book ABAP Objects – ABAP Programming in SAP NetWeaver (2nd edition, by Horst Keller and Sascha Kruger; Galileo Press, 2007, ISBN 978-1-59229-079-6; see sample listing 13.3 starting on page 964) and it fails.

Notice that we have received 2 failure messages associated with this single test method, each one indicating a different airline that had been encountered unexpectedly while testing the assertion. The default for a test method assertion failure is to discontinue executing any remainder of the unit test once an assertion failure is reached, however we have overridden this default behavior by indicating we want the method execution to continue beyond this assertion failure by specifying the parameter “quit = no” on the call to the assertion method.

This version also makes it apparent that our unit test is dependent upon having applicable records existing in transparent table SFLIGHT.

Let’s register in our issues list that this version resolves issue #5, but also introduces new issue #17.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A		No test for code in subroutine adjust_flight_revenue
3	ZAUT101A		No test for code in subroutine apply_flight_discount
4	ZAUT101A		No test for code in subroutine calculate_discounted_airfare
5	**ZAUT101A**	**ZAUT102A**	**No test for code in subroutine get_flights_via_carrier**
6	ZAUT101A		No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A		No test for code in subroutine set_alv_function_module_name
10	ZAUT101A		No test for code in subroutine show_flights
11	ZAUT101A		No test for code in subroutine show_flights_count
12	ZAUT101A		No example of ABAP Unit test for function module
13	ZAUT101A		No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	**ZAUT102A**		**Unit test relies on the presence of records in table SFLIGHT**

4.2 Exercise 8

Program: ZAUT102B

Requirements

Reason for change

- Fix production code bug identified by previous version unit test.

Changes to be applied

1. Change subroutine get_flights_via_carrier to use carrier value provided through subroutine signature.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display for correct carrier with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Action: Specify Airline ‘AA’, no discount, **ALV grid** and press Execute.

Result: Produces **ALV classic list display** for correct carrier with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 2 test methods.

Remarks

With this version we have fixed the bug in the production code and we now see that executing the program produces the expected results in the ALV report, however, we also see that still we are unable to obtain the report in the format of an ALV grid.

Running the unit tests no longer encounters the assertion failures we had seen with the previous version.

4.3 Exercise 9

Program: ZAUT102C

Requirements

Reason for change

- Implement unit test for subroutine set_alv_function_module_name.

Changes to be applied

1. Add new test method set_alv_function_module_name to class tester to call to subroutine set_alv_function_module_name:
   - Add method definition for set_alv_function_module_name to the private section of class tester after the definition for method get_flights_via_carrier:

```
methods      : o
               o
```

```
              o
            , set_alv_function_module_name
              for testing
```

- Include the following method implementation after the implementation for method get_flights_via_carrier:

```
method set_alv_function_module_name.
  constants    : list_flag      type xflag     value space
               , grid_flag      type xflag     value 'X'
               .
  data         : alv_display_function_module
                                type progname
               .
  " The user may select to display the report using alv classic list
  " or alv grid control.  The function modules facilitating these use
  " the same parameter interface and the name of each one contains the
  " string "LIST" or "GRID" respectively.  Here we insure that we
  " get the correct function module name resolved when we provide the
  " flag indicating whether or not to use the grid control.
  perform set_alv_function_module_name using list_flag
                                  changing alv_display_function_module.
  " Here we use the level parameter to indicate that although we may
  " get the incorrect name of the function module based on the selection
  " flag, it is not a critial error (the default for not specifying level).
  cl_abap_unit_assert=>assert_char_cp(
        act                     = alv_display_function_module
        exp                     = '*LIST*'
        msg                     = 'Incorrect ALV program name selected'
        level                   = cl_aunit_assert=>tolerable
        quit                    = cl_aunit_assert=>no
        ).
  perform set_alv_function_module_name using grid_flag
                                    changing alv_display_function_module.
  cl_abap_unit_assert=>assert_char_cp(
        act                     = alv_display_function_module
        exp                     = '*GRID*'
        msg                     = 'Incorrect ALV program name selected'
        level                   = cl_aunit_assert=>tolerable
        quit                    = cl_aunit_assert=>no
        ).
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, **ALV grid** and press Execute.

Result: Produces **ALV classic list** display for correct carrier with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test method set_alv_function_module_name triggers warning; Status message indicates Processed: 1 program, 1 test classes, 3 test methods.

Remarks

With this version we have defined new unit test method set_alv_function_module_name for class tester. The implementation of this method makes two calls to subroutine set_alv_function_module_name, each with a different calling value, performing an assertion on the returned result after each call. Only one of the assertions fails.

Notice also in the ABAP Unit: Results Display report that the assertion failure is marked as a warning. This is because we had indicated to override the default behavior of an assertion failure indicating an error by including

the additional parameter “level = tolerable” on the call to the assertion method. The “level” parameter controls the severity of the failure presented in the ABAP Unit: Results Display report.

Let’s register in our issues list that this version resolves issue #9.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A		No test for code in subroutine adjust_flight_revenue
3	ZAUT101A		No test for code in subroutine apply_flight_discount
4	ZAUT101A		No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A		No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	**ZAUT101A**	**ZAUT102C**	**No test for code in subroutine set_alv_function_module_name**
10	ZAUT101A		No test for code in subroutine show_flights
11	ZAUT101A		No test for code in subroutine show_flights_count
12	ZAUT101A		No example of ABAP Unit test for function module
13	ZAUT101A		No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT

4.4 Exercise 10

Program: ZAUT102D

Requirements

Reason for change

- Fix production code bug identified by previous version unit test.

Changes to be applied

1. In subroutine set_alv_function_module_name, change value of variable alv_grid_function_module from REUSE_ALV_LIST_DISPLAY to REUSE_ALV_GRID_DISPLAY.

Run

Action: Specify Airline ‘AA’, no discount, **ALV grid** and press Execute.

Result: Produces **ALV grid** display for correct carrier with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Action: Specify Airline ‘AA’, **discount 50**, **ALV classic list** and press Execute.

Result: Produces ALV classic list display for correct carrier **with half fares** with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Action: Specify Airline ‘AA’, **discount 100**, ALV **classic list** and press Execute.

Result: Produces ALV classic list display for correct carrier **with free fares** with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 3 test methods.

Remarks

With this version we have fixed the bug in the production code and now are able to get either an ALV list or ALV grid when executing the program. All unit tests now pass.

4.5 Exercise 11

Program: ZAUT102E

Requirements

Reason for change

- Implement unit test for subroutine apply_flight_discount.

Changes to be applied

1. Add new test method apply_flight_discount to class tester to call to subroutine apply_flight_discount:
   - Add method definition for apply_flight_discount to the private section of class tester after the definition for method set_alv_function_module_name:

```
  methods      : o
                 o
                 o
               , apply_flight_discount
                   for testing
```

   - Include the following method implementation after the implementation for method set_alv_function_module_name:

```
method apply_flight_discount.
  constants    : discount_exceeding_100_percent
                               type num03     value 101
               .
  data         : flights_entry  like line
                                  of flights_stack
               .
  " The user may indicate on the initial selection screen to calculate
  " a percentage discount for the airfares to be shown in the report.
  " The selection screen parameter is 3 digits to accept using a 100
  " percent discount (free flight!).  We do not want the discount to
  " be any higher than 100 percent or the airfares will be shown using
  " negative numbers (the airline would pay you to fly!).  Here we
  " insure that the calculated airfare cannot be negative.
  " Set table flights_stack with some records from the sflights table:
  perform get_flights_via_carrier using 'AA'.
  cl_abap_unit_assert=>assert_not_initial(
    act                          = flights_stack
    msg                          = 'No records available for testing flight discount'
    ).
  perform apply_flight_discount using discount_exceeding_100_percent.
  loop at flights_stack
     into flights_entry.
    " We have not specified a quit parameter for the next assertion.
    " The default action is to terminate the test method upon encountering
    " an error.  We do not need to test every record in the table for
    " a negative value since if any one of them is negative then we
    " should expect all of them to be negative.  So we can exit this
    " loop and this test method with the first negative price.  We are
    " using a loop here just in case the first record we encounter had
    " a full price of zero, which would calculate to a discounted price
    " also of zero regardless of an invalid discount value, and would pass
    " the test if we were to inspect only at the first record in the table.
    cl_abap_unit_assert=>assert_equals(
      act                        = flights_entry-price
      exp                        = abs( flights_entry-price )
      msg                        = 'Discounted airfare is negative value'
      ).
  endloop.
endmethod.
```

Run

Action: Specify Airline ‘AA’, **discount 10,** ALV classic list and press Execute.

Result: Produces ALV classic list display for correct carrier with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test method apply_flight_discount triggers failure; Status message indicates Processed: 1 program, 1 test classes, 4 test methods.

Remarks

With this version we have introduced a new unit test method to class tester: apply_flight_discount. It loops through all the rows in global table flights_stack, for each row calling subroutine apply_flight_discount using the row flight price and a discount value deliberately exceeding 100%. The assertion checks that the subroutine ignores the bad discount and that the flight price remains unchanged.

Yet again we see that the addition of a new unit test identifies an assertion failure. The test has identified something wrong with the production code that allows the price of the flight to become negative when a discount is applied.

Let's register in our issues list that this version resolves issue #3.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A		No test for code in subroutine adjust_flight_revenue
3	**ZAUT101A**	**ZAUT102E**	**No test for code in subroutine apply_flight_discount**
4	ZAUT101A		No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A		No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A		No test for code in subroutine show_flights
11	ZAUT101A		No test for code in subroutine show_flights_count
12	ZAUT101A		No example of ABAP Unit test for function module
13	ZAUT101A		No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields

#	Identified	Resolved	Description
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT

4.6 Exercise 12

Program: ZAUT102F

Requirements

Reason for change

- Fix production code bug identified by previous version unit test.

Changes to be applied

1. In subroutine apply_flight_discount, change value of variable percent_100 from 110 to 100.

Run

Action: Specify Airline ‘AA’, **discount 10**, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 4 test methods.

Remarks

With this version we have corrected the production code by setting the value of constant percent_100, defined in subroutine apply_flight_discount, to the correct value of 100. Once again all the unit tests pass.

4.7 Exercise 13

Program: ZAUT102G

Requirements

Reason for change

- Implement unit test for subroutine adjust_flight_revenue.

Changes to be applied

1. Add new test method adjust_flight_revenue to class tester to call to subroutine adjust_flight_revenue:
   - Add method definition for adjust_flight_revenue to the private section of class tester after the definition for method apply_flight_discount:

```
methods      : o
               o
               o
             , adjust_flight_revenue
                 for testing
```

   - Include the following method implementation after the implementation for method apply_flight_discount:

```
method adjust_flight_revenue.
  data         : flights_entry  like line
                                  of flights_stack
               , flight_revenue type flights_row-paymentsum
               .
  " The value of the flight revenue is calculated as the product of the
  " airfare and number of booked seats.  Here we insure that the revenue
  " calculated by the called subroutine represents this product.
  " Set table flights_stack with some records from the sflights table:
  perform get_flights_via_carrier using 'AA'.
  cl_abap_unit_assert=>assert_not_initial(
    act                        = flights_stack
    msg                        = 'No records available for testing flight discount'
    ).
  perform adjust_flight_revenue.
  loop at flights_stack
     into flights_entry.
    flight_revenue             = flights_entry-price * flights_entry-seatsmax.
    cl_abap_unit_assert=>assert_equals(
      act                      = flights_entry-paymentsum
      exp                      = flight_revenue
      msg                      = 'Flight revenue value other than expected'
      ).
  endloop.
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test method adjust_flight_revenue triggers failure; Status message indicates Processed: 1 program, 1 test classes, 5 test methods.

Remarks

With this version we have introduced a new unit test method to class tester: adjust_flight_revenue. It loops through all the rows in global table flights_stack, for each row calling subroutine adjust_flight_revenue and asserting afterward that the flight revenue calculated by this subroutine is the same as the flight price multiplied by the number of seats on the flight.

With another new unit test comes another unit test failure.

Note: By now you might have recognized the pattern we are following here to introduce unit tests into this program: Each new unit test method defined to class tester has the same name as the subroutine it calls. This is not a suggestion to follow this pattern with your own unit test method names, but simply is convenient for the purpose of learning more about automated unit testing. Indeed, as you'll see later, a single unit (subroutine, method, function module, etc.) often has multiple unit test methods calling upon it, so this pattern of naming a unit test method the same as the unit it calls quickly becomes impractical.

Let's register in our issues list that this version resolves issue #2.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	**ZAUT101A**	**ZAUT102G**	**No test for code in subroutine adjust_flight_revenue**
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A		No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A		No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A		No test for code in subroutine show_flights
11	ZAUT101A		No test for code in subroutine show_flights_count
12	ZAUT101A		No example of ABAP Unit test for function module
13	ZAUT101A		No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT

4.8 Exercise 14

Program: ZAUT102H

Requirements

Reason for change

- Fix unit test code bug identified by previous version unit test.

Changes to be applied

1. In ABAP Unit test method adjust_flight_revenue, change assignment of flight_revenue to replace multiplier flights_entry-seatsmax with flights_entry-seatsocc.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 5 test methods.

Remarks

With this version we have corrected *the unit test* causing the failure. Specifically, the formula for calculating the expected flight revenue in unit test method adjust_flight_revenue should use the occupied seats, not the number of seats, to calculate the flight revenue.

After correcting the faulty code of the failing unit test, once again all unit tests pass.

4.9 Exercise 15

Program: ZAUT102I

Requirements

Reason for change

- Implement unit test for subroutine calculate_discounted_airfare.

Changes to be applied

1. Add new test method calculate_discounted_airfare to class tester to call to subroutine calculate_discounted_airfare:

- Add method definition for calculate_discounted_airfare to the private section of class tester after the definition for method adjust_flight_revenue:

```
  methods      : o
                 o
                 o
               , calculate_discounted_airfare
                   for testing
```

- Include the following method implementation after the implementation for method adjust_flight_revenue:

```
method calculate_discounted_airfare.
  constants    : discount_exceeding_100_percent
                                type discount  value 101
               .
  data         : flight_price   type s_price   value '123.45'
               .
  perform calculate_discounted_airfare using flight_price
                                             discount_exceeding_100_percent
                                      changing flight_price
                                             sy-subrc
                                             .
  cl_abap_unit_assert=>assert_equals(
    act                         = flight_price
    exp                         = abs( flight_price )
    msg                         = 'Discounted airfare is negative value'
    ).
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test method calculate_discounted_airfare triggers failure; Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have introduced a new unit test method to class tester: calculate_discounted_airfare. It calls subroutine calculate_discounted_airfare using a random flight price and a discount value deliberately exceeding 100%. The assertion checks that the subroutine ignores the bad discount and that the flight price remains unchanged.

The new unit test method fails. Are you seeing a pattern here? Each time we add a new unit test for a subroutine previously not covered by one we encounter an assertion failure.

Let’s register in our issues list that this version resolves issue #4.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue

#	Identified	Resolved	Description
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	**ZAUT101A**	**ZAUT102I**	**No test for code in subroutine calculate_discounted_airfare**
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A		No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A		No test for code in subroutine show_flights
11	ZAUT101A		No test for code in subroutine show_flights_count
12	ZAUT101A		No example of ABAP Unit test for function module
13	ZAUT101A		No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT

4.10 Exercise 16

Program: ZAUT102J

Requirements

Reason for change

- Fix production code bug identified by previous version unit test.

Changes to be applied

1. In subroutine calculate_discounted_airfare, change value of constants highest_discount_percentage from 110 to 100.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have corrected the production code by setting the value of constant highest_discount_percent, defined in subroutine calculate_discounted_airfare, to the correct value of 100. Once again all the unit tests pass.

4.11 Exercise 17

Program: ZAUT102K

Requirements

Reason for change

- Implement unit test for subroutine get_flight_revenue.

Changes to be applied

1. Add new test method get_flight_revenue to class tester to call to subroutine get_flight_revenue:
   - Add method definition for get_flight_revenue to the private section of class tester after the definition for method calculate_discounted_airfare:

```
  methods      : o
                 o
                 o
               , get_flight_revenue
                   for testing
```

   - Include the following method implementation after the implementation for method calculate_discounted_airfare:

```
method get_flight_revenue.
  data         : flight_price   type s_price
               , flight_booked_seats
                                type s_seatsocc
               , calculated_revenue
                                type s_sum
```

```
              , expected_revenue
                              type s_sum
              .
  flight_price                = 100.
  flight_booked_seats         = 80.
  expected_revenue            = flight_price + flight_booked_seats.
  perform get_flight_revenue using flight_price
                                   flight_booked_seats
                           changing calculated_revenue
                                   .
  cl_abap_unit_assert=>assert_equals(
    act                       = calculated_revenue
    exp                       = expected_revenue
    msg                       = 'Flight revenue value other than expected'
    ).
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test method get_flight_revenue triggers failure; Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we have introduced a new unit test method to class tester: get_flight_revenue. It calls subroutine get_flight_revenue using a random flight price and number of booked seats, presumably expecting to find the flight revenue calculated as the product of these values.

Was there any doubt what we would find by adding a new unit test for a subroutine previously not covered by one?

Let’s register in our issues list that this version resolves issue #6.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	**ZAUT101A**	**ZAUT102K**	**No test for code in subroutine get_flight_revenue**

#	Identified	Resolved	Description
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A		No test for code in subroutine show_flights
11	ZAUT101A		No test for code in subroutine show_flights_count
12	ZAUT101A		No example of ABAP Unit test for function module
13	ZAUT101A		No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT

4.12 Exercise 18

Program: ZAUT102L

Requirements

Reason for change

- Fix unit test code bug identified by previous version unit test.

Changes to be applied

1. In method get_flight_revenue, change assignment of expected_revenue to use multiplication operator instead of addition operator.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we have corrected *the unit test* causing the failure. Specifically, the formula for calculating the expected flight revenue in unit test method get_flight_revenue should use the *product* of the flight price and flight booked seats, not their sum.

After correcting the faulty code of the failing unit test, once again all unit tests pass.

5 ABAP Unit Testing 103 – Writing Unit Tests for Function Modules

This section describes the requirements for the exercise programs associated with the Chapter 6 section titled Implementing Unit Tests for Function Modules in the book Automated Unit Testing with ABAP.

5.1 Exercise 19

Program: ZAUT103A

Requirements

Reason for change

- Demonstrate unit test capability with function module.

Changes to be applied

1. Via SE37, define function group ZAUT:
   - Select from menu: Goto > Function Groups > Create Group
   - Short text: ABAP Unit Test training
   - Package: $TMP
2. Via SE37, define function module ZCALCULATE_DISCOUNTED_AIRFARE:
   - Short text: Calculate discounted airfare
   - Function group ZAUT
   - Attributes tab:
     - Normal function module
     - Start immediately
     - Package: $TMP
   - Import tab:

```
Parameter name    Typing   Associated type
----------------- -------- ---------------
FULL_FARE         TYPE     S_PRICE
DISCOUNT          TYPE     S_DISCOUNT
```

   - Export tab:

```
Parameter name    Typing   Associated type
----------------- -------- ---------------
DISCOUNT_FARE     TYPE     S_PRICE
```

   - Exceptions tab:

```
INVALID_DISCOUNT
```

   - Function executable code to contain the following lines:

```
constants    : highest_discount_percentage
                        type int4      value 110
             .
data         : discount_multiplier
                        type p decimals 3
             .
if discount gt highest_discount_percentage.
  raise invalid_discount.
endif.

discount_multiplier           = ( 100 - discount ) / 100.
discount_fare                 = full_fare * discount_multiplier.
```

   - Include class tester immediately after the ENDFUNCTION statement, to contain the following lines:

```
*=====================================================================
*
*   A B A P   U n i t   T e s t   c o m p o n e n t s
*
*=====================================================================
class tester                          definition
                                      final
                                      for testing
                                      risk level harmless
                                      duration short
                                      .
  private section.
    methods      : calculate_discounted_airfare
                     for testing
                   .
endclass.
class tester                          implementation.
  method calculate_discounted_airfare.
    constants    : discount_exceeding_100_percent
                                  type s_discount
                                                value 101
                   .
    data         : flight_price   type s_price   value '123.45'
                   .
    call function 'ZCALCULATE_DISCOUNTED_AIRFARE'
      exporting
        full_fare                 = flight_price
        discount                  = discount_exceeding_100_percent
      importing
        discount_fare             = flight_price
      exceptions
        others                    = 0
        .
    cl_abap_unit_assert=>assert_equals(
      act                         = flight_price
      exp                         = abs( flight_price )
      msg                         = 'Discounted airfare is negative value'
      ).
  endmethod.
endclass.
```

- Activate all components.

3. In subroutine apply_flight_discount, replace this statement ...

```
  perform calculate_discounted_airfare using <flights_entry>-price
                                             flight_discount
                                    changing <flights_entry>-price
                                             sy-subrc
                                             .
```

… with this statement:

```
  call function 'ZCALCULATE_DISCOUNTED_AIRFARE'
    exporting
      full_fare               = <flights_entry>-price
      discount                = flight_discount
    importing
      discount_fare           = <flights_entry>-price
    exceptions
      others                  = 0
      .
```

Run

Action: Specify Airline ‘AA’, **discount 10**, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test function module ZCALCULATE_DISCOUNTED_AIRFARE (via Function Builder path Function Module > Execute > Unit Tests)

Action: While function module is displayed in the Function Builder editor (SE37), select from menu: Function Module > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test method calculate_discounted_airfare triggers failure; Status message indicates Processed: 1 program, 1 test classes, 1 test methods.

Test program ZAUT103A

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we have created a new function module and new function group for it. In addition, we've changed subroutine apply_flights_discount to call the new function module to provide this service instead of calling the subroutine within our program.

We are now capable of creating a unit test for a function module.

Though for this version of the program all the unit tests pass, the unit test for the function module fails. We will not bother to correct the failing function module unit test since it has no bearing on the correct value the function module is returning to the program.

Note: The main program of a function group is composed of a set of INCLUDEd objects following a prescribed naming convention designated by SAP. For instance, function group Z123 will be composed of main program SAPLZ123, which itself will include a set of objects using the naming convention LZ123xxxx. There will be LZ123TOP to define the global data components for the function module, LZ123UXX to define the various function modules of the function group, and others. The developer may choose to place the various unit tests for a function module into the SAP-designated component intended to hold them, which for function group Z123 would be in objects named LZ123UNITTnn.

Let's register in our issues list that this version resolves issue #12.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue

#	Identified	Resolved	Description
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A		No test for code in subroutine show_flights
11	ZAUT101A		No test for code in subroutine show_flights_count
12	**ZAUT101A**	**ZAUT103A**	**No example of ABAP Unit test for function module**
13	ZAUT101A		No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT

5.2 Exercise 20

Program: ZAUT103B

Requirements

Reason for change

- Discard production and unit test code no longer utilized.

Changes to be applied

1. Remove unused subroutine calculate_discounted_airfare.
2. Remove from class tester test method calculate_discounted_airfare.

Run

Action: Specify Airline ‘AA’, **discount 10**, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we've simply removed dead code from the program.

6 ABAP Unit Testing 104 – Writing Unit Tests for Global Classes

This section describes the requirements for the exercise programs associated with the Chapter 6 section titled Implementing Unit Tests for Global Classes in the book Automated Unit Testing with ABAP.

6.1 Exercise 21

Program: ZAUT104A

Requirements

Reason for change

- Demonstrate unit test capability with global class.

Changes to be applied

1. Via SE24, create global singleton class zcl_flight_revenue_calculator:
   - Description: Flight revenue calculator
   - Inst.Generation: Private
   - Class Type: Usual ABAP Class
   - Final: (checked)
   - Package: $TMP (local object)
2. Via the Attributes tab, define the following static attribute for class zcl_flight_revenue_calculator:

Attribute	Level	Visibility	Read-Only	Typing	Associated Type
SINGLETON	Static Attribute	Public	(blank)	Type Ref To	ZCL_FLIGHT_REVENUE_CALCULATOR

3. Via the Methods tab, define the following static method for class zcl_flight_revenue_calculator:

Method	Level	Visibility	Description
CLASS_CONSTRUCTOR	Static Method	Public	

4. Method CLASS_CONSTRUCTOR is to contain the following source code:

```
method CLASS_CONSTRUCTOR.
  create object singleton.
endmethod.
```

5. Via the Methods tab, define the following instance method for class zcl_flight_revenue_calculator:

Method	Level	Visibility	Description
GET_FLIGHT_REVENUE	Instance Method	Public	Calculate revenue based on number of booked seats

6. Method GET_FLIGHT_REVENUE parameters are to be defined as follows:

Parameter	Type	Pass Value	Optional	Typing Method	Associated Type	Default Value
FARE_PRICE	Importing	unchecked	unchecked	Type	S_PRICE	(blank)
NUMBER_OF_PASSENGERS	Importing	unchecked	unchecked	Type	S_SEATSOCC	(blank)
FLIGHT_REVENUE	Exporting	unchecked	unchecked	Type	S_SUM	(blank)

7. Method GET_FLIGHT_REVENUE is to contain the following source code:

```
  method GET_FLIGHT_REVENUE.
    flight_revenue               = fare_price * number_of_passengers.
  endmethod.
```

8. Activate all components.
9. Return to the screen where "Class" is the first menu selection, then select Utilities > Test Classes > Generate, then (via Wizard):
   - Select Global Class radiobutton
   - Global Class Name: ZCL_FLIGHT_REVENUE_CALCULATOR
   - Test class name: tester (create)
     - Leave all checkboxes unchecked
     - Duration Type: Short
     - Risk Level: Harmless
     - Superclass: (blank)
   - Method: GET_FLIGHT_REVENUE (select)
   - Replace the generated local test class source code with the following:

```
*"* use this source file for your ABAP unit test classes
class tester                              definition
                                          final
                                          for testing
                                          risk level harmless
                                          duration short
                                          .
  private section.
    methods      : get_flight_revenue
                     for testing
                   .
endclass.
class tester                              implementation.
  method get_flight_revenue.
    data         : flight_price   type s_price
                 , flight_booked_seats
                                  type s_seatsocc
                 , calculated_revenue
                                  type s_sum
                 , expected_revenue
                                  type s_sum
                 .
    flight_price                 = 100.
    flight_booked_seats          = 80.
    expected_revenue             = flight_price + flight_booked_seats.
    call method zcl_flight_revenue_calculator=>singleton->get_flight_revenue
      exporting
        fare_price               = flight_price
        number_of_passengers     = flight_booked_seats
      importing
        flight_revenue           = calculated_revenue
        .
    cl_abap_unit_assert=>assert_equals(
      act                        = calculated_revenue
      exp                        = expected_revenue
      msg                        = 'Flight revenue value other than expected'
      ).
  endmethod.
endclass.
```

10. Save and activate all components.
11. Return to the screen where "Class" is the first menu selection, then select Goto > Local Definitions/Implementations > Local Test Classes. Source code of local test class is presented.
12. Return to the screen where "Class" is the first menu selection, then select Class > Run > Unit Tests. This should present the ABAP Unit: Results Display report indicating method get_flight_revenue triggers a failure status message indicates Processed: 1 program, 1 test classes, 1 test methods.

Note: For older releases, the following additional change may be required. From the screen where "Class" is the first menu selection, select menu path Goto > Local Definitions/Implementations > Class relevant local Definitions and include the following lines:

```
class tester definition deferred.
class zcl_flight_revenue_calculator definition local friends tester.
```

13. In subroutine adjust_flight_revenue, replace this statement ...

```
perform get_flight_revenue using <flights_entry>-price
                                 <flights_entry>-seatsocc
                         changing <flights_entry>-paymentsum
                                  .
```

… with this statement:

```
call method zcl_flight_revenue_calculator=>singleton->get_flight_revenue
  exporting
    fare_price              = <flights_entry>-price
    number_of_passengers    = <flights_entry>-seatsocc
  importing
    flight_revenue          = <flights_entry>-paymentsum
    .
```

Run

Action: Specify Airline 'AA', no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test global class zcl_flight_revenue_calculator

Action: From Class Builder: Initial Screen (SE24), select from menu:
Object Type > Run > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicating method get_flight_revenue triggers a failure; Status message indicates Processed: 1 program, 1 test classes, 1 test methods.

Test program ZAUT104A

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have created a new global class. In addition, we've changed subroutine adjust_flight_revenue to call method get_flight_revenue of this new global class to provide this service instead of calling the subroutine within our program.

We are now capable of creating a unit test for a global class.

Though for this version of the program all the unit tests pass, the unit test for the global class fails. We will not bother to correct the failing global class unit test since it has no bearing on the correct value the method of this global class is returning to the program.

Let's register in our issues list that this version resolves issue #13.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen

#	Identified	Resolved	Description
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A		No test for code in subroutine show_flights
11	ZAUT101A		No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	**ZAUT101A**	**ZAUT104A**	**No example of ABAP Unit test for global class**
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT

6.2 Exercise 22

Program: ZAUT104B

Requirements

Reason for change

- Discard production and unit test code no longer utilized.

Changes to be applied

1. Remove unused subroutine get_flight_revenue.
2. Remove from class tester test method get_flight_revenue.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 5 test methods.

Remarks

With this version we’ve simply removed more dead code from the program.

7 ABAP Unit Testing 105 – How Certain ABAP Statements Affect Unit Testing

This section describes the requirements for the exercise programs associated with the Chapter 6 section titled ABAP Statements and Features Affecting Automated Unit Testing in the book Automated Unit Testing with ABAP.

7.1 Exercise 23

Program: ZAUT105A

Requirements

Reason for change

- Implement unit test for subroutine show_flights_count.

Changes to be applied

1. Add new test method show_flights_count to class tester to test call to subroutine show_flights_count:
   - Add method definition for show_flights_count to the private section of class tester after the definition for method adjust_flight_revenue:

```
  methods      : o
                 o
                 o
               , show_flights_count
                   for testing
```

   - Include the following method implementation after the implementation for method adjust_flight_revenue:

```
method show_flights_count.
  constants    : bogus_message_type
                              type symsgty   value '?'
               , bogus_message_id
                              type symsgid   value '?'
               , bogus_message_number
                              type symsgno   value 999
               , bogus_message_variable
                              type symsgv    value '?'
               .
  sy-msgty                    = bogus_message_type.
  sy-msgid                    = bogus_message_id.
  sy-msgno                    = bogus_message_number.
  sy-msgv1                    = bogus_message_variable.
  sy-msgv2                    = bogus_message_variable.
  sy-msgv3                    = bogus_message_variable.
  sy-msgv4                    = bogus_message_variable.
  cl_abap_unit_assert=>assert_equals(
    act                       = sy-msgty
    exp                       = bogus_message_type
    msg                       = 'System field sy-msgty has unexpected value'
    ).
  cl_abap_unit_assert=>assert_equals(
    act                       = sy-msgid
    exp                       = bogus_message_id
    msg                       = 'System field sy-msgid has unexpected value'
    ).
  cl_abap_unit_assert=>assert_equals(
    act                       = sy-msgno
    exp                       = bogus_message_number
    msg                       = 'System field sy-msgno has unexpected value'
    ).
  cl_abap_unit_assert=>assert_equals(
```

```
      act                       = sy-msgv1
      exp                       = bogus_message_variable
      msg                       = 'System field sy-msgv1 has unexpected value'
      ).
    cl_abap_unit_assert=>assert_equals(
      act                       = sy-msgv2
      exp                       = bogus_message_variable
      msg                       = 'System field sy-msgv2 has unexpected value'
      ).
    cl_abap_unit_assert=>assert_equals(
      act                       = sy-msgv3
      exp                       = bogus_message_variable
      msg                       = 'System field sy-msgv3 has unexpected value'
      ).
    cl_abap_unit_assert=>assert_equals(
      act                       = sy-msgv4
      exp                       = bogus_message_variable
      msg                       = 'System field sy-msgv4 has unexpected value'
      ).
    perform show_flights_count.
    cl_abap_unit_assert=>assert_differs(
      act                       = sy-msgty
      exp                       = bogus_message_type
      msg                       = 'System field sy-msgty has unexpected value'
      ).
    cl_abap_unit_assert=>assert_differs(
      act                       = sy-msgid
      exp                       = bogus_message_id
      msg                       = 'System field sy-msgid has unexpected value'
      ).
    cl_abap_unit_assert=>assert_differs(
      act                       = sy-msgno
      exp                       = bogus_message_number
      msg                       = 'System field sy-msgno has unexpected value'
      ).
    cl_abap_unit_assert=>assert_differs(
      act                       = sy-msgv1
      exp                       = bogus_message_variable
      msg                       = 'System field sy-msgv1 has unexpected value'
      ).
    cl_abap_unit_assert=>assert_differs(
      act                       = sy-msgv2
      exp                       = bogus_message_variable
      msg                       = 'System field sy-msgv2 has unexpected value'
      ).
    cl_abap_unit_assert=>assert_differs(
      act                       = sy-msgv3
      exp                       = bogus_message_variable
      msg                       = 'System field sy-msgv3 has unexpected value'
      ).
    cl_abap_unit_assert=>assert_differs(
      act                       = sy-msgv4
      exp                       = bogus_message_variable
      msg                       = 'System field sy-msgv4 has unexpected value'
      ).
  endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods and no cut-issued status message appears.

Remarks

With this version we have introduced a new unit test method to class tester: show_flights_count. It calls subroutine show_flights_count after first setting all of the system variables associated with messages (sy-msgty, sy-msgid, sy-msgno, sy-msgv1 through sy-msgv4) to some bogus value, then asserting afterward that these system variables no longer contain the bogus value, proof that the call to subroutine show_flights_count has caused them to be changed. Notice that the call to subroutine show_flights_count is buried among the setting and asserting of the system variables, making it hard to understand what this unit test is doing. As written, this new unit test exudes the smell "Obscure Test", one of the unit testing smells cataloged by Gerard Meszaros (xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 250).

Since this test executes a subroutine that is issuing a status message via the ABAP MESSAGE statement, we might have expected to see the status message appear during the test, but we do not. This may be due to the fact that the test runner of the Automated Unit Test Framework issues its own status message after having completed running all the tests and we are simply seeing the final one of all the status messages issued.

Let's register in our issues list that this version resolves issue #11.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A		No test for code in subroutine show_flights
11	**ZAUT101A**	**ZAUT105A**	**No test for code in subroutine show_flights_count**
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle

#	Identified	Resolved	Description
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT

7.2 Exercise 24

Program: ZAUT105B

Requirements

Reason for change

- Simplify the unit test code for testing subroutine show_flights_count.

Changes to be applied

1. Refactor test method show_flights_count of class tester:
   - Move the 4 constants to follow the private section header of the class definition.
   - Define in the private section the following new methods without the "for testing" clause, and their respective empty methods in the implementation section, after the definition for method show_flights_count:
     - set_bogus_message
     - assert_message_is_bogus
     - assert_message_not_bogus
   - Move all the statements of method show_flights_count setting fields of structure sy to method set_bogus_message.
   - Move all the assert_equals assertions of method show_flights_count to method assert_message_is_bogus.
   - Move all the assert_differs assertions of method show_flights_count to method assert_message_not_bogus.
   - In method show_flights_count, precede the perform statement with calls to methods set_bogus_message and assert_message_is_bogus.
   - In method show_flights_count, succeed the perform statement with a call to method assert_message_not_bogus.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods and no cut-issued status message appears.

Remarks

With this version we have simplified the implementation of unit test method show_flights_count by introducing 3 additional helper methods – the first sets the system variables associated with messages to bogus values, the second asserts that these system variables are set to bogus values and the third asserts that these system variables are not set to bogus values. Now, with only 3 method calls and a perform statement, the implementation of unit test method show_flights_count is much easier to understand.

Here we have defined new private methods for class tester but they are not themselves test methods – none of them has a "for testing" clause in its method definition. Indeed, methods defined within a unit test class that do not contain the "for testing" clause are regarded as "Test Utility Methods", a unit testing pattern cataloged by Gerard Meszaros (xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 599). These methods simply provide services to be called by other methods of the same unit test class. As you can see, it makes identifying and understanding the intent behind unit test method show_flights_count much easier. Accordingly, this unit test method no longer exudes the smell "Obscure Test" (xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 250).

7.3 Exercise 25

Program: ZAUT105C

Requirements

Reason for change

- Force a unit test failure to confirm previous refactoring works correctly.

Changes to be applied

1. In test method show_flights_count of class tester, confirm assertions can catch error by commenting out call to subroutine show_flights_count.

Run

Action: Specify Airline 'AA', no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test method show_flights_count triggers failure; Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have changed unit test method show_flights_count by commenting out the call to subroutine show_flights_count. In doing so, we simply are confirming that a unit test will fail when it encounters a situation where its assertion should fail.

7.4 Exercise 26

Program: ZAUT105D

Requirements

Reason for change

- Determine how a message statement with severity "information" affects unit test.

Changes to be applied

1. Uncomment the call to subroutine show_flights_count in test method show_flights_count.
2. Change subroutine show_flights_count message severity to ‘I’.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Information message appearing in popup window shows number of flights conforming to selection criteria followed by ALV classic list display appearing after pressing enter.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods and no cut-issued information message appears.

Remarks

With this version we have again changed unit test method show_flights_count by reactivating the statement commented out in the previous version. Also, we have raised the severity of the ABAP MESSAGE statement issued by subroutine show_flights_count from status to information, one which we expect should cause the message to appear in an information popup window when issued. We see that this occurs when we run the program in its production mode, but we do not see the message when the unit tests are run.

Since this test executes a subroutine that is issuing an information message via a message statement, we might have expected to see that information message appear during the test, but we do not. Unlike with the final status message we had encountered before with version ZAUT105A, this result now makes it appear that the test runner of the Automated Unit Test Framework intercepts and suppresses such messages.

7.5 Exercise 27

Program: ZAUT105E

Requirements

Reason for change

- Determine how a message statement with severity "warning" affects unit test.

Changes to be applied

1. Change subroutine show_flights_count message severity to ‘W’.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Error message appears at bottom left of screen showing number of flights conforming to selection criteria, discontinuing program execution after pressing enter.

Note: This is expected behavior of a warning message, which when encountered during the start-of-selection event will have its severity level elevated from warning to error.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods and no cut-issued warning message appears.

Remarks

With this version we have again raised the severity of the ABAP MESSAGE statement issued by subroutine show_flights_count from information to warning.

Since this test executes a subroutine that is issuing a warning message via the ABAP MESSAGE statement, we might have expected to see that warning message appear during the test, but we do not. A warning message will behave differently depending on the classic ABAP event block from which it is issued. When issued from the at selection-screen event, it will appear as a warning message on the screen, but when issued from the start-of-selection classic ABAP event block then it will appear as an error message. Accordingly, since the tests are not being called from any of the classic ABAP event blocks in the program, we might now conclude that the test runner of the Automated Unit Test Framework, which is the caller of these tests, has a similar effect on the execution of message statements, perhaps one that indicates status, information and warning messages are simply to be suppressed for the duration of the unit test.

7.6 Exercise 28

Program: ZAUT105F

Requirements

Reason for change

- Determine how a message statement with severity "error" affects unit test.

Changes to be applied

1. Change subroutine show_flights_count message severity to ‘E’.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Error message appears at bottom left of screen showing number of flights conforming to selection criteria, discontinuing program execution after pressing enter.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test method show_flights_count failed with message Exception Error <CX_AUNIT_UNCAUGHT_MESSAGE>; Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have again raised the severity of the ABAP MESSAGE statement issued by subroutine show_flights_count from warning to error.

Since we have been steadily increasing the severity level of the message we know to be issued by the subroutine called by the unit test, we might have expected that at some point we would use a severity that would behave differently than simply suppressing the message. It is with severity “error” that we finally see this difference – the unit test run fails and the ABAP Unit: Results Display report is presented showing that unit test method show_flights_count failed with an Exception Error.

Let’s register in our issues list that this version introduces new issue #18.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A		No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count

#	Identified	Resolved	Description
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT
18	**ZAUT105F**		**Unit test issues Exception Error upon encountering MESSAGE statement with severity E**

7.7 Exercise 29

Program: ZAUT105G

Requirements

Reason for change

- Determine how a message statement with severity "abort" affects unit test.

Changes to be applied

1. Change subroutine show_flights_count message severity to ‘A’.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Cancel popup window appears showing number of flights conforming to selection criteria, discontinuing program execution after pressing enter.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test method show_flights_count failed with message Exception Error <CX_AUNIT_UNCAUGHT_MESSAGE> (same as for ZAUT105F); Status message indicates Processed: 1 program, 1 test classes, 6 test methods

Remarks

With this version we have again raised the severity of the ABAP MESSAGE statement issued by subroutine show_flights_count from error to abort.

After setting to the next higher level the severity of the message we know to be issued by the subroutine called by the unit test, again the unit test run fails and we see similar results to those we saw with the previous version.

Let's register in our issues list that this version introduces new issue #19.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A		No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT

#	Identified	Resolved	Description
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	**ZAUT105G**		**Unit test issues Exception Error upon encountering MESSAGE statement with severity A**

7.8 Exercise 30

Program: ZAUT105H

Requirements

Reason for change

- Determine how a message statement with severity "exit" affects unit test.

Changes to be applied

1. Change subroutine show_flights_count message severity to ‘X’.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Runtime Error - Description of Exception report screen appears – otherwise known as the short dump screen – providing details about the exception encountered.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test method show_flights_count failed with message Runtime Error <MESSAGE_TYPE_X>; Status message indicates Processed: 1 program, 1 test classes, **0 test methods**.

Remarks

With this version we have again raised the severity of the ABAP MESSAGE statement issued by subroutine show_flights_count from abort to exit.

After setting to the highest level the severity of the message we know to be issued by the subroutine called by the unit test, again the unit test run fails and we see similar results to those we saw with the previous two versions.

At this point we know the following about how the test runner of the ABAP Unit Test Framework will respond to messages issued by the message statement:

Message severity	Description of severity	Behavior by test runner of the ABAP Unit Test Framework when encountered
S	status	Message is not detectable

I	information	Message does not pop up as it would during normal foreground execution
W	warning	Message is not detectable and does not interfere with ABAP Unit Test running to completion
E	error	ABAP Unit: Results Display report appears showing Exception Error <CX_AUNIT_UNCAUGHT_MESSAGE>
A	abort	ABAP Unit: Results Display report appears showing Exception Error <CX_AUNIT_UNCAUGHT_MESSAGE>
X	exit	ABAP Unit: Results Display report appears showing Runtime Error <MESSAGE_TYPE_X>

Let's register in our issues list that this version introduces new issue #20.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A		No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle

#	Identified	Resolved	Description
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	**ZAUT105H**		**Unit test issues Runtime Error upon encountering MESSAGE statement with severity X**

7.9 Exercise 31

Program: ZAUT105I

Requirements

Reason for change

- Attempt to intercept CX_AUNIT_UNCAUGHT_MESSAGE exception within unit test.

Changes to be applied

1. Change subroutine show_flights_count message severity back to ‘E’.
2. Add try-catch block around the call from method show_flights_count to subroutine show_flights_count, catching class-based exception CX_AUNIT_UNCAUGHT_MESSAGE and issuing a message via call to cl_abap_unit_assert=>fail:

```
try.
  perform show_flights_count.
catch cx_aunit_uncaught_message.
  cl_abap_unit_assert=>fail(
    msg                      = 'Caught exception in test method show_flights_count'
    ).
endtry.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Error message appears at bottom left of screen showing number of flights conforming to selection criteria, discontinuing program execution after pressing enter.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test method show_flights_count failed with message Exception Error <CX_AUNIT_UNCAUGHT_MESSAGE>, **despite having a catch clause for exactly this class-based exception**; Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have reduced the severity of the ABAP MESSAGE statement issued by subroutine show_flights_count from exit back down to error, which, as we saw with version ZAUT105F, should cause unit test method show_flights_count to fail for having raised Exception Error CX_AUNIT_UNCAUGHT_MESSAGE. Also, we've changed unit test method show_flights_count by surrounding the call to subroutine show_flights_count with a try-endtry block to catch the CX_AUNIT_UNCAUGHT_MESSAGE exception.

The unit test fails again with test method show_flights_count raising the Exception Error CX_AUNIT_UNCAUGHT_MESSAGE, just as it did with version ZAUT105F. Accordingly, it seems the test runner of the ABAP Unit Test Framework does not recognize our attempt to intercept such an exception within the unit test code.

7.10 Exercise 32

Program: ZAUT105J

Requirements

Reason for change

- Confirm CX_AUNIT_UNCAUGHT_MESSAGE exception can be caught within unit test.

Changes to be applied

1. Within method show_flights_count, raise the class-based exception CX_AUNIT_UNCAUGHT_MESSAGE in the try-catch block prior to calling subroutine show_flights_count:

```
try.
  raise exception type cx_aunit_uncaught_message.
  perform show_flights_count.
catch cx_aunit_uncaught_message.
  cl_abap_unit_assert=>fail(
    msg                      = 'Caught exception in test method show_flights_count'
    ).
endtry.
```

Run

Action: Specify Airline 'AA', no discount, ALV classic list and press Execute.

Result: Error message appears at bottom left of screen showing number of flights conforming to selection criteria, discontinuing program execution after pressing enter.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test method show_flights_count failed with same message text as that used for the msg parameter of the call to method cl_abap_unit_assert=>fail from the catch

clause of the try block, proving that the catch clause is capable of intercepting this class-based exception; Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have changed unit test method show_flights_count by deliberately raising the exception CX_AUNIT_UNCAUGHT_MESSAGE within the try-endtry block, just to see whether we are able to catch the CX_AUNIT_UNCAUGHT_MESSAGE exception.

The unit test still fails, but we have confirmed that a unit test method is capable of intercepting such an exception within the unit test code. The conclusion we can draw here is that the test runner of the ABAP Unit Test Framework also intercepts this class-based exception during its own processing but does not allow the exception to be propagated back to the try block established in the test method. **This means that messages issued with severity error, abort and exit appearing in the executable code will present challenges to running clean tests if such messages were to be encountered during an ABAP Unit test execution.**

7.11 Exercise 33

Program: ZAUT105K

Requirements

Reason for change

- Reset code to issue status message in subroutine show_flights_count.

Changes to be applied

1. Change subroutine show_flights_count message severity back to ‘S’.
2. Remove the raise exception statement in test method show_flights_count.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Error message appears at bottom left of screen showing number of flights conforming to selection criteria, discontinuing program execution after pressing enter.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods and no cut-issued status message appears.

Remarks

With this version we have changed unit test method show_flights_count by removing the statement deliberately raising exception CX_AUNIT_UNCAUGHT_MESSAGE within the try-endtry block and by resetting the message severity in subroutine show_flights_count from error back to its original value of status.

Now that we have restored the program to a state where it no longer encounters a message statement of a severity causing unit test failure, once again the unit tests pass.

7.12 Exercise 34

Program: ZAUT105L

Requirements

Reason for change

- Implement unit test for subroutine show_flights.

Changes to be applied

1. Add new test method show_flights to class tester to test call to subroutine show_flights for the following 3 carriers: LH, UA, AA:
   - Add method definition for show_flights to the private section of class tester after the definition for method assert_message_not_bogus:

```
  methods      : o
                 o
                 o
               , show_flights
                   for testing
```

   - Include the following method implementation after the implementation for method assert_message_not_bogus:

```
method show_flights.
  constants    : lufthansa      type s_carr_id value 'LH'
               , united_airlines
                                type s_carr_id value 'UA'
               , american_airlines
                                type s_carr_id value 'AA'
               .
  data         : carrier_id_stack
                                type table
                                  of s_carr_id
               , carrier_id_entry
                                like line
                                  of carrier_id_stack
               .
  append: lufthansa             to carrier_id_stack
        , united_airlines       to carrier_id_stack
        , american_airlines     to carrier_id_stack
        .
  loop at carrier_id_stack
     into carrier_id_entry.
    perform get_flights_via_carrier using carrier_id_entry.
    perform show_flights using 00
                               abap_false.
  endloop.
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Error message appears at bottom left of screen showing number of flights conforming to selection criteria, discontinuing program execution after pressing enter.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears for flights of carrier 'LH';
Press back, exit, cancel or ESCape, then ALV classic list appears for flights of carrier 'UA';
Press back, exit, cancel or ESCape, then ALV classic list appears for flights of carrier 'AA';
Press back, exit, cancel or ESCape, then returns to editor with status message indicating Processed: 1 program, 1 test classes, 7 test methods.

Action: Again: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears for flights of carrier 'LH';
Wait at least 2 minutes before pressing back, exit, cancel or ESCape, then ALV classic list appears for flights of carrier 'UA';
Press back, exit, cancel or ESCape, then ALV classic list appears for flights of carrier 'AA';
Press back, exit, cancel or ESCape, then ABAP Unit: Results Display report appears indicating test class tester triggers warning about excessive execution time; Status message indicates Processed: 1 program, 1 test classes, 7 test methods

Remarks

With this version we have introduced a new unit test method to class tester: show_flights. It loops through in internal table containing the 3 airline codes representing Lufthansa, United Airlines and American Airlines, then for each one does the following:

- Calls subroutine get_flights_via_carrier using the associated airline code.
- Calls subroutine show_flights using parameter values indicating no flight discount and no ALV grid list.

Indeed, the code of its implementation is simply a copy of the implementation code for unit test method get_flights_via_carrier and changed slightly to apply to calling subroutine show_flights.

All the unit tests pass, but not before we are presented with a series of 3 ALV classic reports of flights, each of which requiring us to issue a command to allow the unit test to continue with its execution.

This version introduces two more issues for us:

1. Identical constants for Lufthansa, United Airlines and American Airlines now appear in two different methods of class tester, an example “Cut-and-Past Code Reuse” and one of the causes of the unit test smell known as "Test Code Duplication" cataloged by Gerard Meszaros (xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 213).
2. Running the unit test requires user intervention for all the tests to run to completion, an example of a unit test exuding the smell “Manual Intervention”, one of the unit testing smells cataloged by Gerard Meszaros (xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 250).

The first issue is minor, but the second one presents a serious impediment to our ability to write unit tests that can run to completion without the need for user intervention. We will address both of these issues in subsequent exercises.

Let’s register in our issues list that this version resolves issue #10, but it also introduces new issues #21 and #22.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare

#	Identified	Resolved	Description
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	**ZAUT101A**	**ZAUT105L**	**No test for code in subroutine show_flights**
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	**ZAUT105L**		**Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester**
22	**ZAUT105L**		**ALV report presented during unit test requires user intervention to continue test to completion**

7.13 Exercise 35

Program: ZAUT105M

Requirements

Reason for change

- Disable presentation of ALV report during unit test.

Changes to be applied

1. Comment out the "for testing" clause on method definition show_flights of class tester.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Error message appears at bottom left of screen showing number of flights conforming to selection criteria, discontinuing program execution after pressing enter.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods and no cut-issued status message appears.

Remarks

With this version we have removed the “for testing” clause from unit test show_flights of class tester. This removes the necessity for the user to interact with the unit test to allow it to run to completion, but it also introduces an issue we had resolved once already – again there is no active unit test for subroutine show_flights.

Let’s register in our issues list that this version introduces new issue #23, with a reference to issue #10 that had been resolved already for this same reason.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue

#	Identified	Resolved	Description
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L		Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	**ZAUT105M**		**Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)**

7.14 Exercise 36

Program: ZAUT105N

Requirements

Reason for change

- Determine the effect a WRITE statement has upon unit testing.

Changes to be applied

1. Clone the message statement appearing in subroutine show_flights_count and convert it into a corresponding write statement to be placed immediately after the message statement:

```
write: /            flights_count
     ,              'flights are available for carrier' ##NO_TEXT
     ,              carrier
     .
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria; Pressing Back, Exit, Cancel or ESCape causes content of write statement to appear in a classic list report screen. Pressing Back, Exit, Cancel or ESCape results in returning to the initial selection screen.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Content of write statement appears on the screen but is preceded by red and green highlighted rows indicating the following:

```
+------------------------------------------------------------------------------
|Internal Session for Isolated Test Class Execution
+------------------------------------------------------------------------------
|Warning
|=======
|Program: <program name>
|Class:   <test class name>
|This window is displayed because your test case has triggered
|a list command like follows:
|- new page
|- leave to list processing
|- uline
|- write
|- new page
|- ...
|This as any interactive technique is not permitted !!
+------------------------------------------------------------------------------
|Please avoid the use of these statements. To locate them in the source code
|setting break-points on the mentioned statements should help.
+------------------------------------------------------------------------------
|       <content of write statement appears here>
```

Pressing Back, Exit, Cancel or ESCape results in a status message indicating Processed: 1 program, 1 test classes, 6 test methods and no cut-issued status message appears.

Remarks

Version ZAUT105L demonstrated to us the result when a unit test encounters production code that creates and presents an ALV report. With this version we experimented further with report output to determine what happens when a unit test encounters production code presenting a report via simple ABAP WRITE statements.

Here we have inserted a single ABAP WRITE statement into subroutine show_flights-count, a statement to report the same message content presented by the ABAP MESSAGE statement in this subroutine, simply to determine how such a statement behaves during a unit test.

The unit tests still pass, but we are presented with a new issue for us to consider: The test runner presents the full screen Internal Session for Isolated Test Class Execution list upon encountering a classical list statement, followed by the report content. Not only that, but because this full screen is presented the unit test now requires user interaction to allow it to run to completion, another example of a unit test exuding the smell “Manual Intervention” (xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 250).

At this point we have learned that there are some ABAP statements incompatible with using ABAP Unit testing. Statements such as those noted in the full page presented during the unit test facilitate producing what is referred to as a "classical list", an obsolete technique for producing report content. The book "Official ABAP Programming Guidelines" (Keller, Thummel, 2010, SAP Press) states in the explanation for "Rule 5.20: Use the SAP List Viewer" that classical lists should no longer be used, and that any dynpro-based output should be facilitated through the use of the SAP List Viewer (ALV).

If we are trying to retrofit automated unit tests into an existing program containing such reporting statements, then we will need to decide whether we are prepared to replace these reporting statements with corresponding ALV reports. An alternative is to retain these reporting statements and simply refactor the program, if necessary, to the extent that these statements are confined to subroutines for which we will not write unit tests, and accept the fact that the new unit tests do not provide full coverage of the program. We will revisit this topic in a subsequent exercise program.

Let’s register in our issues list that this version introduces new issue #24.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name

#	Identified	Resolved	Description
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A		Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L		Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M		Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	**ZAUT105N**		**Unit test presents full screen warning upon encountering any list command (write, uline, etc.)**

7.15 Exercise 37

Program: ZAUT105O

Requirements

Reason for change

- Undo the changes implemented in the previous version.

Changes to be applied

1. Simply copy version ZAUT105M forward to this version.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria; Pressing Back, Exit, Cancel or ESCape results in returning to the initial selection screen.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods and no cut-issued status message appears.

Remarks

With this version we’ve simply removed the changes we made with the prevision version.

Since the WRITE statement introduced into the program in the previous version was not there already, we have lost nothing by restoring the program back to the image of version ZAUT105M, but we have gained insight about the effect such reporting statements have upon unit testing. This will be explored further in subsequent exercise programs.

8 ABAP Unit Testing 106 – How Unit Testing Enables Confident Refactoring

This section describes the requirements for the exercise programs associated with the Chapter 6 section titled How Automated Unit Testing Enables Confident Refactoring in the book Automated Unit Testing with ABAP.

8.1 Exercise 38

Program: ZAUT106A

Requirements

Reason for change

- Refactor production code so that subroutines perform only the actions associated with their names.

Changes to be applied

1. Move the perform statements to subroutines apply_flight_discount and adjust_flight_revenue (and their comments) from subroutine show_flights to subroutine present_report, ahead of the call to subroutine show_flights_count, changing the name of the parameter on the call to subroutine apply_flight_discount from flights_discount to discount:

```
form present_report using discount
                            type discount
                          via_grid
                            type xflag.
    " Adjust flights fare by specified discount:
    perform apply_flight_discount using discount.
    " Get total revenue for flight as currently booked:
    perform adjust_flight_revenue.
    perform show_flights_count.
    perform show_flights using via_grid.
endform.
```

2. Remove parameter flight_discount from definition of subroutine show_flights.
3. Remove first parameter on all calls to subroutine show_flights.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods and no cut-issued status message appears.

Remarks

With this version we have moved the calls to subroutines apply_flight_discount and adjust_flight_revenue from subroutine show_flights to subroutine present_report. This was done to allow subroutine show_flights to have

only those responsibilities suggested by its name, which certainly should not include applying flight discounts and adjusting flight revenue to the rows of flights it will show to the user.

Changes such as this allow a unit of code to adhere to the *Single Responsibility Principle*, which represents the "S" among the five SOLID principles defined and promoted by Robert C. Martin for making software designs more flexible, understandable and maintainable:

> **Single Responsibility Principle** – A [software unit] should only have a single responsibility, that is, only changes to one part of the software's specification should be able to affect the specification of the [software unit].

How, you may be wondering, did the previous version of subroutine present_report run afoul of the Single Responsibility Principle? It did so because it was responsible for performing the following tasks:

1. applying a discount to the flight price
2. adjusting the flight revenue
3. showing a report of the flight rows

The process of applying a discount to the fight price and of adjusting the flight revenue certainly are tasks that need to be performed so that their respective results are reflected in the report, but a subroutine named show_report is not the correct place to perform these tasks. Each of the activities listed above has its corresponding specification for how the activity is designed and is to perform, but a change to only one of these specifications should cause this subroutine to be affected. With the previous version, this subroutine could have been affected by a change to any of the tasks listed above.

Running the ABAP Unit tests still show no errors after we have altered the program slightly. This again reinforces for us that in the process of changing code we have not broken anything that already was working. We can now begin to see the benefit of having automated unit tests for a program – in virtually no time at all we are able to confirm that nothing in this program has been broken despite introducing a simple change, or perhaps even massive changes, into the program.

Let's register in our issues list that this version resolves issue #14.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog

#	Identified	Resolved	Description
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	**ZAUT101A**	**ZAUT106A**	**Subroutine show_flights violates the single responsibility principle**
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L		Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M		Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)

9 ABAP Unit Testing 107 – Diagnosing the Absence of Sufficient Test Data

This section describes the requirements for the exercise programs associated with the Chapter 6 section titled Diagnosing the Absence of Sufficient Test Data in the book Automated Unit Testing with ABAP.

9.1 Exercise 39

Program: ZAUT107A

Requirements

Reason for change

- Expose the absence of applicable test data in table SFLIGHT.

Changes to be applied

1. In method get_flights_via_carrier of class tester:
   - Change value of all carrier constants to have first character '?'.
   - Replace the following set of statements ...

```
concatenate 'Selection of'
            carrier_id_entry
            'gives different airlines'
       into failure_message separated by space.
perform get_flights_via_carrier using carrier_id_entry.
```

… with this set of statements:

```
" Confirm applicable test records exist in this environment:
concatenate 'No records found for carrier'
            carrier_id_entry
            'in environment'
            sy-sysid
            sy-mandt
       into failure_message separated by space.
perform get_flights_via_carrier using carrier_id_entry.
cl_abap_unit_assert=>assert_not_initial(
  act                       = flights_stack
  msg                       = failure_message
  level                     = cl_aunit_assert=>tolerable
  quit                      = cl_aunit_assert=>no
  ).
concatenate 'Selection of'
            carrier_id_entry
            'gives different airlines'
       into failure_message separated by space.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test method get_flights_via_carrier triggers warning with associated message for each carrier that no records can be found in this environment; Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have changed unit test method get_flights_via_carrier of class tester to assert that its call to subroutine get_flights_via_carrier for a given carrier code results in records populating global table flights_stack, since a carrier code resulting in an empty table would allow the unit test to pass for that carrier. Furthermore, we have deliberately changed the values of the three airline carrier codes to guarantee that global table flights_stack always will be empty after calling subroutine get_flights_via_carrier.

The unit test fails with 3 warning messages indicating the lack of records for the carrier in the environment in which the unit test is being run. This is what we would expect after deliberately changing the carrier codes to values for which we do not expect to find any records.

Unlike many other languages for which automated unit tests can be written, the ABAP language provides specific CRUD statements (Create, Retrieve, Update, Delete) to interact with the underlying database tables. So far some of our unit tests have been relying on the presence of records in table SFLIGHT retrieved by the unit test to fulfill a successful assertion. Here we see the result of running the unit test when we deliberately request records that we expect not to exist in the database tables. The dependency of using an actual database to provide records during unit testing is a topic to be covered in subsequent exercise programs.

9.2 Exercise 40

Program: ZAUT107B

Requirements

Reason for change

- Reset to retrieve applicable test data from table SFLIGHT or to identify its absence.

Changes to be applied

1. In method get_flights_via_carrier of class tester, change value of all carrier constants back to their former values (‘LH’, ‘UA’, ‘AA’).

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have changed unit test method get_flights_via_carrier of class tester to reinstated the values of the three airline carrier codes back to the values they had prior to the previous version.

Once again, all the unit tests pass.

If you do not get this test result, then it would indicate that there are no records in your environment for at least one of the associated carriers. Follow the instructions provided in section **1.5 Insuring test data records exist** to create such records so that you do get this test result.

10 ABAP Unit Testing 108 – Creating and Using Fabricated Test Data

This section describes the requirements for the exercise programs associated with the Chapter 6 section titled Creating and Using Fabricated Test Data in the book Automated Unit Testing with ABAP.

10.1 Exercise 41

Program: ZAUT108A

Requirements

Reason for change

- Begin the process of having the unit test use a cache of fabricated test data.

Changes to be applied

1. Add new test method get_test_flights_via_carrier to class tester:
   - Add method definition for get_test_flights_via_carrier to the private section of class tester after the definition for method show_flights:

```
methods      : o
               o
               o
             , get_test_flights_via_carrier
                 importing
                   carrier
                     type carrier
                 changing
                   flights_stack
                     type flights_list
                   flights_count
                     type int4
```

   - Include the following method implementation after the implementation for method show_flights:

```
method get_test_flights_via_carrier.
  clear flights_stack.
  describe table flights_stack lines flights_count.
endmethod.
```

2. At all locations within class tester where a call is made to subroutine get_flights_via_carrier:
   - Set field carrier with the carrier value used to call subroutine get_flights_via_carrier.
   - Replace the call call to subroutine get_flights_via_carrier with a comparable call to method get_test_flights_via_carrier:

```
  call method get_test_flights_via_carrier
    exporting
      carrier                = carrier
    changing
      flights_stack          = flights_stack
      flights_count          = flights_count
      .
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates methods adjust_flight_revenue and apply_flight_discount of class tester trigger failures and method get_flights_via_carrier of class tester triggers warnings; Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have introduced a new method to class tester: get_test_flights_via_carrier. It is not marked with the "for testing" clause, but is intended to provide test records for flights so we are not dependent on such records existing in table SFLIGHT. Its implementation is empty for now. In addition, we've changed all the calls within the unit test methods to now call method get_test_flights_via_carrier instead of performing subroutine get_flights_via_carrier as they had been doing. This means that during the unit test run all flights records will be supplied to the unit test methods via method get_test_flights_via_carrier.

The unit test methods are failing because we no longer are using records from table SFLIGHT but are calling new method get_test_flights_via_carrier to provide test data, and this method is providing an empty table of flights. Also with this version we no longer have a unit test for subroutine get_flights_via_carrier.

Let's register in our issues list that this version introduces new issue #25.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count

#	Identified	Resolved	Description
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A		Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L		Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M		Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	**ZAUT108A**		**No longer any test for subroutine get_flights_via_carrier**

10.2 Exercise 42

Program: ZAUT108B

Requirements

Reason for change

- Continue the process of having the unit test use a cache of fabricated test data.

Changes to be applied

1. Add to class tester private static attribute named test_flights_stack, located after the constants statement and defined as follows:

```
  class-data   : test_flights_stack
                                type flights_list
               .
```

2. Change method get_test_flights_via_carrier to loop at static attribute test_flights_stack to fill records in global variable flights_stack when variable carrier has a non-blank value:

```
method get_test_flights_via_carrier.
  data         : test_flights_entry
                                like line
                                  of test_flights_stack
               .
  clear flights_stack.
  if carrier is not initial.
    loop at test_flights_stack
       into test_flights_entry
      where carrid              eq carrier.
      append test_flights_entry
          to flights_stack.
    endloop.
  endif.
  describe table flights_stack lines flights_count.
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates methods adjust_flight_revenue and apply_flight_discount of class tester trigger failures and method get_flights_via_carrier of class tester triggers warnings; Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have introduced a static attribute to class tester: test_flights_stack. In addition, we have provided formerly empty unit test method get_test_flights_via_carrier with an implementation to copy from internal table attribute test_flights_stack to global table flights_stack those rows associated with the specified carrier.

The unit test still fails because although method get_test_flights_via_carrier copies records from new internal table test_fights_stack to global table flights_stack, internal table test_fights_stack is empty, so again no records are provided for the unit test to use.

10.3 Exercise 43

Program: ZAUT108C

Requirements

Reason for change

- Complete the process of having the unit test use a cache of fabricated test data.

Changes to be applied

1. Add static method class_setup to class tester. It is to create 5 records in static attribute test_flights_stack with a unique value for flight date and occupied seats for each of 2 unique connections for each of the three carriers LH, UA and AA. Unique flight dates are to start with the current date and increment with each new record; occupied seats are to be 10 fewer with each new date for the same connection; connection id is to start at 01 and increment with each new set of 5 flight dates; and constant values shared between all records are to be the following:
   - price - 1000
   - currency - USD
   - plane type - 747-400
   - maximum seats – 385

   payment sum is to be number of occupied seats multiplied by price.

   - Add static method definition for class_setup to the private section of class tester after the class-data statement:

```
  class-methods: class_setup
              .
```

   - Include the following method implementation at the top of the class implementation component for class tester:

```
method class_setup.
  constants    : lufthansa      type s_carr_id value 'LH'
               , united_airlines
                                type s_carr_id value 'UA'
               , american_airlines
                                type s_carr_id value 'AA'
               .
  data         : carrier_id_stack
                                type table
                                  of s_carr_id
               , carrier_id_entry
                                like line
                                  of carrier_id_stack
               , test_flights_entry
                                like line
                                  of test_flights_stack
               .
  append: lufthansa             to carrier_id_stack
        , united_airlines       to carrier_id_stack
        , american_airlines     to carrier_id_stack
        .
  test_flights_entry-mandt      = sy-mandt.
  test_flights_entry-fldate     = sy-datum.
  test_flights_entry-price      = 1000.
  test_flights_entry-currency   = 'USD'.
  test_flights_entry-planetype  = '747-400'.
  test_flights_entry-seatsmax   = 385.
  loop at carrier_id_stack
     into carrier_id_entry.
    test_flights_entry-carrid   = carrier_id_entry.
    do 02 times.
      add 01 to test_flights_entry-connid.
      do 05 times.
        add 01 to test_flights_entry-fldate.
        test_flights_entry-seatsocc
                                = test_flights_entry-seatsmax - sy-index * 10.
        test_flights_entry-paymentsum
                                = test_flights_entry-price
                                * test_flights_entry-seatsocc.
        append test_flights_entry
            to test_flights_stack.
      enddo.
    enddo.
```

```
    endloop.
  endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have introduced the use of the class_setup method. This method of a test class is invoked automatically by the test runner of the Automated Unit Test Framework before any of the unit tests defined for the test class. It is similar in nature to a class_constructor method defined for a class, meaning it is invoked once and only once prior to any methods being called.

Here the class_setup method generates the test data records to be populated into internal table static attribute test_fights_stack. Placing this activity in the class_setup method is convenient because we only need to perform this activity once on behalf of all the unit test methods.

The unit tests now pass because there are now internally-generated test records available for each of the unit test methods to use. At this point we have freed ourselves from the dependence on using test records retrieved from table SFLIGHT.

Let’s register in our issues list that this version resolves issue #17.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog

#	Identified	Resolved	Description
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	**ZAUT102A**	**ZAUT108C**	**Unit test relies on the presence of records in table SFLIGHT**
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L		Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M		Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A		No longer any test for subroutine get_flights_via_carrier

10.4 Exercise 44

Program: ZAUT108D

Requirements

Reason for change

- Allow unit test to present ALV report showing fabricated test data used by the unit test.

Changes to be applied

1. Uncomment the "for testing" clause of test method show_flights of class tester.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears for flights of carrier 'LH'; Press back, exit, cancel or ESCape, then
ALV classic list appears for flights of carrier 'UA'; Press back, exit, cancel or ESCape, then
ALV classic list appears for flights of carrier 'AA'; Press back, exit, cancel or ESCape, then
Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we have reactivated the “for testing” clause of unit test method show_flights of class tester. This affords us the opportunity to examine the test records created by method class_setup of class tester.

Let’s register in our issues list that this version resolves issue #23.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog

#	Identified	Resolved	Description
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L		Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	**ZAUT105M**	**ZAUT108D**	**Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)**
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A		No longer any test for subroutine get_flights_via_carrier

10.5 Exercise 45

Program: ZAUT108E

Requirements

Reason for change

- Introduce use of setup method in test class.

Changes to be applied

1. Add empty instance method setup to test class:
   - Add method definition for setup to the private section of class tester after the definition for method get_test_flights_via_carrier:

```
methods      : o
               o
               o
             , setup
```

   - Include the following method implementation after the implementation for method class_setup:

```
method setup.
endmethod.
```

2. Remove from methods apply_flight_discount and adjust_flight_revenue of class tester those statements setting parameter carrier to explicit value 'AA' along with subsequent call to test method get_test_flights_via_carrier.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears for flights of carrier 'LH'; Press back, exit, cancel or ESCape, then
ALV classic list appears for flights of carrier 'UA'; Press back, exit, cancel or ESCape, then
ALV classic list appears for flights of carrier 'AA'; Press back, exit, cancel or ESCape, then
ABAP Unit: Results Display report indicates class tester triggers failures during methods adjust_flight_revenue and apply_flight_discount; Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we have introduced the use of the setup method. This method of a test class is invoked automatically by the test runner of the Automated Unit Test Framework before each unit test method is called. Here the setup method is intended to replace the explicit calls in test methods apply_flight_discount and adjust_flight_revenue to retrieve test records for carrier “AA”.

The unit tests fail because we neglected to provide any code in the setup method to perform the activity of calling method get_test_flights_via_carrier for carrier “AA” as had been done explicitly by both unit test methods apply_flight_discount and adjust_flight_revenue.

10.6 Exercise 46

Program: ZAUT108F

Requirements

Reason for change

- Fix the unit test failures encountered in the previous version.

Changes to be applied

1. In empty instance method setup of test class, add statements setting parameter carrier to explicit value 'AA' followed by call to test method get_test_flights_via_carrier (code that had been removed from previous version):

```
method setup.
  carrier                     = 'AA'.
  call method get_test_flights_via_carrier
    exporting
      carrier                 = carrier
    changing
      flights_stack           = flights_stack
      flights_count           = flights_count
      .
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears for flights of carrier 'LH'; Press back, exit, cancel or ESCape, then
ALV classic list appears for flights of carrier 'UA'; Press back, exit, cancel or ESCape, then
ALV classic list appears for flights of carrier 'AA'; Press back, exit, cancel or ESCape, then
Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we have provided the setup method with the same code formerly embedded in unit test methods apply_flight_discount and adjust_flight_revenue.

Once again, all unit tests pass. Notice we did not directly invoke method setup of class tester, yet we see evidence that it was invoked now that the unit tests are passing.

10.7 Exercise 47

Program: ZAUT108G

Requirements

Reason for change

- Remove the presentation of ALV reports from unit test.

Changes to be applied

1. Again comment out "for testing" clause of method show_flights.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have again commented out the “for testing” clause of unit test show_flights. This version no longer requires manual intervention to allow the unit tests to run to completion, but again it introduces an issue we had resolved once already – there is no active unit test for subroutine show_flights.

Let’s register in our issues list that this version introduces new issue #26, with a reference to issue #23 that had been resolved already for this same reason.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name

#	Identified	Resolved	Description
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L		Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A		No longer any test for subroutine get_flights_via_carrier
26	**ZAUT108G**		**Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)**

10.8 Exercise 48

Program: ZAUT108H

Requirements

Reason for change

- Introduce use of teardown method in test class.

Changes to be applied

1. Add instance method teardown that checks table flights_stack is empty:
   - Add method definition for teardown to the private section of class tester after the definition for method setup:

```
methods      : o
               o
               o
             , teardown
```

   - Include the following method implementation after the implementation for method setup:

```
method teardown.
  cl_abap_unit_assert=>assert_initial(
    act                         = flights_stack
    msg                         = 'Teardown method finds table flights_stack is not empty'
    ).
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class triggers failures for teardown method; Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have introduced the use of a teardown method. The logical opposite of the setup method, the teardown method of a test class is invoked automatically by the test runner of the Automated Unit Test Framework *after* each unit test method is called. We have provided the teardown method with an implementation to assert that the flights_stack table we’ve been using to provide test data to the unit test methods is empty.

The unit test fails in the new teardown method because we neglected to clear table flights_stack at the completion of any of the unit tests.

10.9 Exercise 49

Program: ZAUT108I

Requirements

Reason for change

- Fix the unit test failures encountered in the previous version.

Changes to be applied

1. Include in method teardown a statement to clear table flights_stack prior to the assertion.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have cleared table flights_stack in the teardown method itself.

Once again, all unit tests pass.

10.10 Exercise 50

Program: ZAUT108J

Requirements

Reason for change

- Introduce use of class_teardown method in test class.

Changes to be applied

1. Add static method class_teardown that checks table test_flights_stack is empty.
   - Add static method definition for class_teardown to the private section of class tester:

```
class-methods: o
               o
               o
             , class_teardown
```

   - Include the following method implementation after the implementation for method teardown:

```
method class_teardown.
  cl_abap_unit_assert=>assert_initial(
    act                         = test_flights_stack
    msg                         = 'Class_teardown method finds table test_flights_stack is not empty'
    ).
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class triggers failures for class_teardown method; Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have introduced the use of a class_teardown method. The logical opposite of the class_setup method, the class_teardown method of a test class is invoked automatically by the test runner of the Automated Unit Test Framework *after* the last unit test method has completed. We have provided the class_teardown method with an implementation to assert that the test_flights_stack table is now empty.

The unit test fails in the new class_teardown method because table test_flights_stack remains filled with records after the last unit test has completed.

10.11 Exercise 51

Program: ZAUT108K

Requirements

Reason for change

- Fix the unit test failures encountered in the previous version.

Changes to be applied

1. Include in method class_teardown a statement to clear table test_flights_stack prior to the assertion.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have cleared table test_flights_stack in the class_teardown method itself.

Once again, all unit tests pass.

10.12 Exercise 52

Program: ZAUT108L

Requirements

Reason for change

- Eliminate duplication of unit test code.

Changes to be applied

1. Move constants defining carriers Lufthansa, United Airlines and American Airlines from method class_setup to private section of class.
2. In method setup, replace assignment of value 'AA' to carrier to use constant for American Airlines.
3. Discard statements defining airline constants from methods get_flights_via_carrier and show_flights.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we removed the duplication of carrier constants across multiple methods of class tester, making them members of the unit test class itself instead of defining them within the unit test methods in which they formerly appeared, and also took the opportunity to remove the use of the airline carrier code literal “AA” used in method setup and replaced it with a reference to its counterpart carrier constant.

This reinforces for us that during the process of changing code we have not broken anything that already was working. Again we see the benefit of having automated unit tests for a program – in virtually no time at all we are able to confirm that nothing in this program has been broken despite introducing a simple change into the program, this time introducing the change into the unit test code rather than into the production code.

Let’s register in our issues list that this version resolves issue #21.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount

#	Identified	Resolved	Description
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A		Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	**ZAUT105L**	**ZAUT108L**	**Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester**
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L;

#	Identified	Resolved	Description
			see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A		No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G		Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)

11 ABAP Unit Testing 109 – Gaining Control Over References to Modifiable Global Variables Within Subroutines

This section describes the requirements for the exercise programs associated with the Chapter 6 section titled Gaining Control Over References to Modifiable Global Variables Within Subroutines in the book Automated Unit Testing with ABAP.

11.1 Exercise 53

Program: ZAUT109A

Requirements

Reason for change

- Modify the program so that subroutine get_flights_via_carrier no longer refers directly to any modifiable global variables.

Changes to be applied

1. Apply the following changes to subroutines get_flights_via_carrier and its callers:
   - Add parameters to subroutine get_flights_via_carrier:

```
changing flights_stack
           type flights_list
         flights_count
           type int4.
```

   - Change call to subroutine get_flights_via_carrier in classic event block at selection-screen to include parameters:

```
changing flights_stack
         flights_count.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have changed the signature of subroutine get_flights_via_carrier so that its code refers to subroutine signature parameters instead of directly to modifiable global variables. Callers to this subroutine were changed accordingly to provide the required parameters. This is the first of the six subroutines referring directly to modifiable global variables. Removing direct references to modifiable global variables from within subroutines is a first step to gaining control over the use of global variables.

11.2 Exercise 54

Program: ZAUT109B

Requirements

Reason for change

- Modify the program so that subroutine apply_flight_discount no longer refers directly to any modifiable global variables.

Changes to be applied

1. Apply the following changes to subroutines apply_flight_discount and its callers:
   - Add parameters to subroutine apply_flight_discount:

```
changing flights_stack
           type flights_list.
```

   - In subroutine apply_flight_discount, change definition of <flights_entry> to "like line of flights_stack".
   - Change call to subroutine apply_flight_discount from subroutine present_report to include parameters:

```
changing flights_stack.
```

   - Change call to subroutine apply_flight_discount from method apply_flight_discount of class tester to include parameters:

```
changing flights_stack.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have changed the signature of subroutine apply_flight_discount so that its code refers to subroutine signature parameters instead of directly to modifiable global variables. Callers to this subroutine were changed accordingly to provide the required parameters.

11.3 Exercise 55

Program: ZAUT109C

Requirements

Reason for change

- Modify the program so that subroutine adjust_flight_revenue no longer refers directly to any modifiable global variables.

Changes to be applied

1. Apply the following changes to subroutines adjust_flight_revenue and its callers:
   - Add parameters to subroutine adjust_flight_revenue:

```
changing flights_stack
           type flights_list.
```

   - In subroutine adjust_flight_revenue, change definition of <flights_entry> to "like line of flights_stack".
   - Change call to subroutine adjust_flight_revenue from subroutine present_report to include parameters:

```
changing flights_stack.
```

   - Change call to subroutine adjust_flight_revenue from method adjust_flight_revenue of class tester to include parameters:

```
changing flights_stack.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have changed the signature of subroutine adjust_flight_revenue so that its code refers to subroutine signature parameters instead of directly to modifiable global variables. Callers to this subroutine were changed accordingly to provide the required parameters.

11.4 Exercise 56

Program: ZAUT109D

Requirements

Reason for change

- Modify the program so that subroutine show_flights_count no longer refers directly to any modifiable global variables.

Changes to be applied

1. Apply the following changes to subroutines show_flights_count and its callers:
   - Add parameters to subroutine show_flights_count:

```
using flights_count
        type int4
      carrier
        type carrier.
```

   - Change call to subroutine show_flights_count from subroutine present_report to include parameters:

```
 using flights_count
       carrier.
```

   - Change call to subroutine show_flights_count from method show_flights_count of class tester to include parameters:

```
 using flights_count
       carrier.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have changed the signature of subroutine show_flights_count so that its code refers to subroutine signature parameters instead of directly to modifiable global variables. Callers to this subroutine were changed accordingly to provide the required parameters.

11.5 Exercise 57

Program: ZAUT109E

Requirements

Reason for change

- Modify the program so that subroutine show_flights no longer refers directly to any modifiable global variables.

Changes to be applied

1. Apply the following changes to subroutines show_flights and its callers:
   - Add parameters to subroutine show_flights:

```
changing flights_stack
           type flights_list.
```

   - Change call to subroutine show_flights from subroutine present_report to include parameters:

```
       changing flights_stack.
```

   - Change call to subroutine show_flights from method show_flights of class tester to include parameters:

```
       changing flights_stack.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have changed the signature of subroutine show_flights so that its code refers to subroutine signature parameters instead of directly to modifiable global variables. Callers to this subroutine were changed accordingly to provide the required parameters.

11.6 Exercise 58

Program: ZAUT109F

Requirements

Reason for change

- Modify the program so that subroutine present_report no longer refers directly to any modifiable global variables.

Changes to be applied

1. Apply the following changes to subroutines present_report and its callers:
   - Add parameters to subroutine present_report:

```
          carrier
            type carrier
          flights_count
            type int4
changing flights_stack
           type flights_list.
```

- Change call to subroutine present_report in classic event block end-of-selection to include parameters:

```
          carrier
          flights_count
changing flights_stack.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have changed the signature of subroutine present_report so that its code refers to subroutine signature parameters instead of directly to modifiable global variables. Callers to this subroutine were changed accordingly to provide the required parameters. This was the last of the six subroutines referring directly to modifiable global variables.

Let’s register in our issues list that this version resolves issue #16.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights

#	Identified	Resolved	Description
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	**ZAUT101A**	**ZAUT109F**	**Subroutines directly access global data not defined as constant**
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A		No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G		Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)

12 ABAP Unit Testing 201 – Gaining Control Over Unit Test Coverage of Input

This section describes the requirements for the exercise programs associated with the Chapter 7 section titled Encapsulating Indirect Input Processes to Accommodate Unit Testing in the book Automated Unit Testing with ABAP.

12.1 Exercise 59

Program: ZAUT201A

Requirements

Reason for change

- Refactoring: Migrate flights retrieval processing into a singleton class.

Changes to be applied

1. Extract flights-related retrieval processing into singleton class flights_organizer; include within the public section the same definitions as those for global declarations for:
   types flights_row; flights_list; carrier
   constants flights_table_name
   data flights_stack
   - Define the following class ahead of the global fields:

```
*=====================================================================
*
*   O O   C l a s s e s
*
*=====================================================================
class flights_organizer               definition
                                      final
                                      create private
                                      .
  public section.
    types        : flights_row    type sflight
                 , flights_list   type standard table
                                    of flights_row
                 , carrier        type s_carr_id
                 .
    constants    : flights_table_name
                                  type tabname   value 'SFLIGHT'
                 .
    data         : flights_stack  type flights_list
                                    read-only
                 .
    class-methods: class_constructor
                 , get_instance
                     returning
                       value(instance)
                         type ref
                           to flights_organizer
                 .
    methods      : get_flights_via_carrier
                     importing
                       carrier
                         type carrier
                 .
  private section.
    class-data   : singleton      type ref
                                    to flights_organizer
                 .
endclass.
class flights_organizer               implementation.
```

```
  method class_constructor.
    create object singleton.
  endmethod.
  method get_instance.
    instance                    = singleton.
  endmethod.
  method get_flights_via_carrier.
    clear flights_stack.
    if carrier is not initial.
      try.
        select *
          into table flights_stack
          from (flights_table_name)
         where carrid              eq carrier
          .
      catch cx_root ##NO_HANDLER ##CATCH_ALL.
        " Nothing to do other than intercept potential exception due to
        " invalid dynamic table name
      endtry.
    endif.
  endmethod.
endclass.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have created the first local class for use by the production code. It is not yet being used, but simply is defined. Its name is flights_organizer and it effectively encapsulates all program code relating to the retrieval and organization of flights records. Notice that this new class defines public type statements for flights_row, flights_list and carrier and a public constant flights_table_name, duplicating their global counterpart definitions already defined within the program. Notice also that all the unit tests still pass, confirming that its introduction into the program does not cause any of the existing unit tests to fail. Also notice that its method get_flights_via_carrier has the same flights record retrieval code as found in subroutine get_flights_via_carrier.

This new class is defined as a singleton class. A singleton class is one where only one instance of this type of class will exist during the execution of the code. The aspects of this class that make it a singleton class are these:

(a) The “create private” clause appearing on the class definition statement, which means an instance of this class can be created only by this class.

(b) The fact that the class_constructor method of this class is the only place within this class where an instance of this class is created.

The combination of these aspects means that the class_constructor method will create the sole instance and that sole instance will be provided to any external callers requesting an instance via the get_instance method.

Singleton classes come with their own baggage rendering them undesirable entities in some situations. In a subsequent exercise we will eliminate the singleton nature of this class.

Let’s register in our issues list that this version introduces new issue #27.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A		Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with

#	Identified	Resolved	Description
			severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A		No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G		Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	**ZAUT201A**		**First singleton class is introduced: flights_organizer**

12.2 Exercise 60

Program: ZAUT201B

Requirements

Reason for change

- Remove global types and constants now managed by flights retrieval singleton class.

Changes to be applied

1. Eliminate the following global definitions:
   types flights_row; flights_list; carrier
   constants flights_table_name
2. Convert references to them to instead reference their counterpart definitions in class flights_organizer. For instance, replace line ...:

```
parameters      :   carrier     type carrier obligatory
```

… with the line ...

```
parameters      :   carrier     type flights_organizer=>carrier obligatory
```

… and replace line …

```
        from (flights_table_name)
```

… with the line:

```
        from (flights_organizer=>flights_table_name)
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With the previous version we had introduced new local class flights_organizer that provided public types and constants duplicating some global types and constants found in the program. Now that these types and constants are defined in a class that encapsulates information about organizing flights, there no longer is a need to retain their global counterparts, so they are removed with this version and references to the global definitions are updated to refer to the public definitions of class flights_organizer. We are doing this in an effort to remove from the program all global variables as per “Rule 6.3: Do Not Declare Global Variables” defined in the book "Official ABAP Programming Guidelines" (Keller, Thummel, 2010, SAP Press).

12.3 Exercise 61

Program: ZAUT201C

Requirements

Reason for change

- Remove global data field flights_stack.

Changes to be applied

1. Move the definition for global field flights_stack to follow the class-data definition for test_flights_stack in class tester:

```
class-data   : test_flights_stack
                            type flights_organizer=>flights_list
             .
data         : flights_stack  type flights_organizer=>flights_list
             .
```

2. Change subroutine get_flights_via_carrier to make calls to methods of class flights_organizer to obtain flights content:

```
form get_flights_via_carrier using carrier
                                   type flights_organizer=>carrier
                          changing flights_count
                                   type int4.
    data         : flights_organizer
                                type ref
                                  to flights_organizer
                 .
    call method flights_organizer=>get_instance
      receiving
        instance                = flights_organizer.
    call method flights_organizer->get_flights_via_carrier
```

```
      exporting
        carrier                  = carrier.
    describe table flights_organizer->flights_stack lines flights_count.
endform.
```

3. In classic event block at selection-screen, change call to subroutine get_flights_via_carrier to remove extraneous parameter flights_stack.
4. Change subroutine present_report to make calls to methods of class flights_organizer to obtain flights content:

```
form present_report using discount
                          type discount
                        via_grid
                          type xflag
                        carrier
                          type flights_organizer=>carrier
                        flights_count
                          type int4.
    data         : flights_organizer
                                type ref
                                  to flights_organizer
                 , flights_stack  type flights_organizer=>flights_list
                 .
    call method flights_organizer=>get_instance
      receiving
        instance                 = flights_organizer.
    flights_stack                = flights_organizer->flights_stack.
    " Adjust flights fare by specified discount:
    perform apply_flight_discount using discount
                                changing flights_stack.
    " Get total revenue for flight as currently booked:
    perform adjust_flight_revenue changing flights_stack.
    perform show_flights_count using flights_count
                                     carrier.
    perform show_flights using via_grid
                       changing flights_stack.
endform.
```

5. In classic event block end-of-selection, change call to subroutine present_report to remove extraneous parameter flights_stack.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have removed the first of the two global variables defined in this program – flights_stack. Notice that its removal as a global variable required this definition be defined as an attribute of unit test class tester so that the unit tests could continue to reference a variable by this name. Notice also that now subroutine get_flights_via_carrier no longer retrieves records directly from table SFLIGHT but instead calls the methods of class flights_organizer to provide flight records via its public attribute flights_stack.

12.4 Exercise 62

Program: ZAUT201D

Requirements

Reason for change

- Make singleton instance of class flights_organizer publicly accessible.

Changes to be applied

1. Change definition component of class flights_organizer:
   - Move singleton attribute from private to public section, to follow constant flights_table_name, and apply read-only clause to it.
   - Remove definition and implementation of static method get_instance.
   - Remove the empty private section header from class flights_organizer.

   Afterward, definition component of class flights_organizer should look like this:

```
class flights_organizer               definition
                                      final
                                      create private
                                      .
  public section.
    types        : flights_row    type sflight
                 , flights_list   type standard table
                                    of flights_row
                 , carrier        type s_carr_id
                 .
    constants    : flights_table_name
                                  type tabname   value 'SFLIGHT'
                 .
    class-data   : singleton      type ref
                                    to flights_organizer
                                      read-only
                 .
    data         : flights_stack  type flights_list
                                      read-only
                 .
    class-methods: class_constructor
                 .
    methods      : get_flights_via_carrier
                     importing
                       carrier
                         type carrier
                 .
endclass.
```

2. Apply the following changes to subroutine get_flights_via_carrier:
   - Remove reference variable flights_organizer.
   - Remove call to static method get_instance of class flights_organizer.
   - Change call to method get_flights_via_carrier of class flights_organizer to access this method through its singleton instance:

```
call method flights_organizer=>singleton->get_flights_via_carrier ...
```

   - Change the describe table statement to reference directly the singleton attribute of class flights_organizer:

```
describe table flights_organizer=>singleton->flights_stack lines flights_count.
```

   Afterward, subroutine get_flights_via_carrier should look like this:

```
form get_flights_via_carrier using carrier
```

```
                                  type flights_organizer=>carrier
                          changing flights_count
                                  type int4.
    call method flights_organizer=>singleton->get_flights_via_carrier
      exporting
        carrier                = carrier.
    describe table flights_organizer=>singleton->flights_stack lines flights_count.
endform.
```

3. Apply the following changes to subroutine present_report:
   - Remove reference variable flights_organizer.
   - Remove call to static method get_instance of class flights_organizer.
   - Change reference to attribute flights_stack of class flights_organizer to access this attribute through its singleton instance:

```
    flights_stack                = flights_organizer=>singleton->flights_stack.
```

Afterward, subroutine present_report should look like this:

```
form present_report using discount
                          type discount
                        via_grid
                          type xflag
                        carrier
                          type flights_organizer=>carrier
                        flights_count
                          type int4.
    data         : flights_stack  type flights_organizer=>flights_list
                 .
    flights_stack                = flights_organizer=>singleton->flights_stack.
    " Adjust flights fare by specified discount:
    perform apply_flight_discount using discount
                                changing flights_stack.
    " Get total revenue for flight as currently booked:
    perform adjust_flight_revenue changing flights_stack.
    perform show_flights_count using flights_count
                                     carrier.
    perform show_flights using via_grid
                       changing flights_stack.
endform.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have enabled external entities to access the singleton instance of class flights_organizer directly, in read-only mode. Accordingly, there no longer is a reason for this class to have a public static method to provide this instance to external entities, so method get_instance has been removed. Those locations in the code previously calling method get_instance of class flights_organizer have been modified to reference its singleton attribute directly and remove local variables defined for the purpose of receiving the reference to this singleton returned by method get_instance.

The class is still a singleton class. The only difference now is that the attribute containing the singleton instance is directly accessible by external entities, obviating the need for the public static method through which external entities formerly obtained this instance.

12.5 Exercise 63

Program: ZAUT201E

Requirements

Reason for change

- Remove global data field flights_count.

Changes to be applied

1. Define a new counter type to class flights_organizer, after the definition for type carrier:

```
types       : o
              o
              o
            , counter       type int4
```

2. Add new functional method get_flights_count to class flights_organizer:
   - Add method definition for get_flights_count to the public section of class flights_organizer after the definition for method get_flights_via_carrier:

```
methods     : o
              o
              o
            , get_flights_count
                returning
                  value(flights_count)
                    type counter
```

   - Include the following method implementation after the implementation for method get_flights_via_carrier:

```
method get_flights_count.
  describe table flights_stack lines flights_count.
endmethod.
```

3. Remove global field flights_count.
4. Apply the following changes to subroutine get_flights_via_carrier:
   - Remove the changing parameter flights_count
   - Remove the describe table statement

   Afterward it should look like this:

```
form get_flights_via_carrier using carrier
                                   type flights_organizer=>carrier.
   call method flights_organizer=>singleton->get_flights_via_carrier
     exporting
       carrier                  = carrier.
endform.
```

5. Remove the extraneous changing parameter from callers of subroutine get_flights_via_carrier.
6. Apply the following changes to subroutine present_report:
   - Remove the using parameter flights_count
   - Define a new data field: flights_count type flights_organizer=>counter

- Prior to calling subroutine show_flights_count, use the functional call format to call method get_flights_count of class flights_organizer:

```
    flights_count                = flights_organizer=>singleton->get_flights_count( ).
```

Afterward it should look like this:

```
form present_report using discount
                          type discount
                        via_grid
                          type xflag
                        carrier
                          type flights_organizer=>carrier.
    data         : flights_stack  type flights_organizer=>flights_list
                 , flights_count  type flights_organizer=>counter
                 .
    flights_stack                = flights_organizer=>singleton->flights_stack.
    " Adjust flights fare by specified discount:
    perform apply_flight_discount using discount
                             changing flights_stack.
    " Get total revenue for flight as currently booked:
    perform adjust_flight_revenue changing flights_stack.
    flights_count                = flights_organizer=>singleton->get_flights_count( ).
    perform show_flights_count using flights_count
                                     carrier.
    perform show_flights using via_grid
                      changing flights_stack.
endform.
```

7. In classic event end-of-selection, remove the extraneous using parameter flights_count from call to subroutine present_report.
8. In classic event at selection-screen,
   - Replace this statement ...

```
    if flights_count le 00.
```

... with this statement:

```
    if flights_organizer=>singleton->get_flights_count( ) le 00.
```

9. In private section of class tester, define new data field: flights_count type flights_organizer=>counter:

```
    data         : o
                   o
                   o
                 , flights_count  type flights_organizer=>counter
                 .
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have removed the last of the two global variables defined in this program – flights_count. Notice that its removal as a global variable required this definition be defined as an attribute of unit test class tester so that the unit tests could continue to reference a variable by this name. Notice also that now class flights_organizer provides a public method to retrieve the number of flights. Accordingly, it no longer is necessary for the subroutine signatures to provide a parameter referencing flights_count since this value is now available via a call to public method get_flights_count of singleton class flights_organizer.

Let's register in our issues list that this version resolves issue #15.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	**ZAUT101A**	**ZAUT201E**	**Program defines global data fields**
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT

#	Identified	Resolved	Description
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A		No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G		Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer

12.6 Exercise 64

Program: ZAUT201F

Requirements

Reason for change

- Refactoring: Remove unnecessary subroutine get_flights_via_carrier.

Changes to be applied

1. Now that subroutine get_flights_via_carrier has only one caller and it only calls flights_organizer=>singleton->get_flights, eliminate subroutine get_flights_via_carrier, moving the call it contains to the location where the subroutine is called. Notice that with the elimination of subroutine get_flights_via_carrier there is no syntax error with method get_flights_via_carrier of class tester because it has not been calling this subroutine since program ZAUT108A.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

This version eliminates the call from the “at selection-screen” classic event block to subroutine get_flights_via_carrier, which had been reduced by refactoring only to calling method get_flights_via_carrier of class flights_organizer. Now the “at selection-screen” classic event block calls method get_flights_via_carrier of class flights_organizer directly. Notice that with the elimination of subroutine get_flights_via_carrier there is no syntax error with unit test method get_flights_via_carrier of class tester because it has not been calling this subroutine since program ZAUT108A.

At this point the only call to the method flights=>singleton->get_flights_via_carrier is from the “at selection-screen” classic event block, so flights=>singleton->flights_stack will remain empty for the duration of running unit tests.

Let’s register in our issues list that this version affects issue #25, but also introduces new issue #28.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module

#	Identified	Resolved	Description
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	**ZAUT108A**	**(ZAUT201F)**	**No longer any test for subroutine get_flights_via_carrier**
26	ZAUT108G		Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	**ZAUT201F**		**Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)**

12.7 Exercise 65

Program: ZAUT201G

Requirements

Reason for change

- In class tester, use instance of flights_organizer to provide flights records.

Changes to be applied

1. In test method get_flights_via_carrier of class tester, change the call to method assert_not_initial of class cl_abap_unit_assert after the call to method get_test_flights_via_carrier, replacing the "act" parameter value flights_stack with value flights_organizer=>singleton->flights_stack.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test method get_flights_via_carrier class triggers 3 warnings; Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version the unit test method get_flights_via_carrier uses the flights_organizer instance to provide the set of flights. The unit tests fail because the attribute flights_stack of the instance of flights_organizer contains no records, as it would during execution of the program in production mode.

12.8 Exercise 66

Program: ZAUT201H

Requirements

Reason for change

- Fix the unit test warnings encountered in the previous version.

Changes to be applied

1. Include the statement "class tester definition deferred" ahead of the definition statement for class flights_organizer:

```
class tester definition deferred.
```

2. Apply the friends clause to the class statement of flights_organized to grant friendship to class tester:

```
class flights_organizer              definition
                                     o
                                     o
                                     friends tester
                                     .
```

3. In test method get_flights_via_carrier, place the following statement after the call to method get_test_flights_via_carrier:

```
    flights_organizer=>singleton->flights_stack
                              = flights_stack.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we provide records to the attribute flights_stack of the instance of flights_organizer, allowing the unit test to pass. Notice that we needed to change the definition of class flights_organizer to indicate that it now has “friend” class tester, a class that flights_organizer trusts to be able to see and change all of its private attributes. In this case, unit test method get_flights_via_carrier updates the attribute flights_stack of the instance of flights_organizer by populating it with test flights records created by method get_test_flights_via_carrier of class tester. It is the “friends” clause of the flights_organizer class definition statement that grants permission to an external entity to change its private attributes. In this case the external entity is method get_flights_via_carrier of class tester and the private attribute is flights_stack of the instance of flights_organizer.

This shows an example of “Back Door Manipulation”, a unit testing pattern cataloged by Gerard Meszaros (see xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 327), where an external entity manipulates the state of a component before that component is used in a unit test.

12.9 Exercise 67

Program: ZAUT201I

Requirements

Reason for change

- Provide class tester change access to class flights_organizer indirectly via empty interface.

Changes to be applied

1. Define empty interface flights_organizer_testable ahead of class flights_organizer:

```
*=====================================================================
*
*   0 0   I n t e r f a c e s
*
*=====================================================================
interface flights_organizer_testable.
endinterface.
```

2. Remove the statement "class tester definition deferred".

3. Change the friends clause of class flights_organizer to grant friendship to interface flights_organizer_testable instead of class tester:

```
class flights_organizer               definition
                                      o
                                      o
                                      friends flights_organizer_testable
                                      .
```

4. Change class tester to include a reference to interface flights_organizer_testable in its public section, to be placed ahead of the private section header:

```
  public section.
    interfaces   : flights_organizer_testable
                   .
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

The previous version enabled methods of class tester to change the private attributes of the instance of class flights_organizer. That version used an explicit class named on the “friends” clause of the class definition for flights_organizer. With this version we have generalized this association via an empty interface called flights_organizer_testable. Now the flights_organizer class definition statement indicates that its friends are all those classes that implement the flights_organizer_testable interface. Only class tester implements this interface, so, by association, class tester is a friend of class flights_organizer and is granted permission to changes its private attributes.

The end result is the same as with the previous version, but this shows how we can avoid designating specific classes as friends to other classes and simply provide an interface name that other classes can implement in order to be considered friends of the class offering the friendship.

The following benefits become available between classes tester and flights_organizer by using indirect friendship through an interface:

1. Class flights_organizer in productive code has no reference to a class that is not regarded as productive code
2. Class tester may now be renamed and/or divided into multiple classes during refactoring and it will not require any corresponding changes to the friends clause of class flights_organizer

12.10 Exercise 68

Program: ZAUT201J

Requirements

Reason for change

- Remove extraneous attribute flights_stack from class tester.

Changes to be applied

1. Remove attribute flights_stack from class tester.
2. Change any references in class tester to attribute flights_stack to now reference attribute flights_organizer=>singleton->flights_stack.
3. In method get_flights_via_carrier of class tester, remove the following statement:

```
flights_organizer=>singleton->flights_stack
                            = flights_stack.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have removed the attribute flights_stack from class tester and instead rely on the public attribute flights_stack of class flights_organizer to supply flight records.

12.11 Exercise 69

Program: ZAUT201K

Requirements

Reason for change

- Confirm test data is being used by class tester.

Changes to be applied

1. Uncomment the "for testing" clause of method show_flights.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears for flights of carrier 'LH'; Press back, exit, cancel or ESCape, then
ALV classic list appears for flights of carrier 'UA'; Press back, exit, cancel or ESCape, then
ALV classic list appears for flights of carrier 'AA'; Press back, exit, cancel or ESCape, then
Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

This version simply enables us to confirm that the test data created by class tester is being used with the unit tests.

Let's register in our issues list that this version resolves issue #26.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle

#	Identified	Resolved	Description
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	**ZAUT108G**	**ZAUT201K**	**Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)**
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F		Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)

12.12 Exercise 70

Program: ZAUT201L

Requirements

Reason for change

- Remove extraneous attribute flights_count from class tester.

Changes to be applied

1. Again comment out the "for testing" clause of method show_flights.

2. Change test class tester to remove its flights_count attribute.
3. Remove parameter flights_count from test method get_test_flights_via_carrier.
4. Change all callers to method get_test_flights_via_carrier to remove the extraneous flights_count parameter.
5. Change method show_flights_count of class tester as follows:
   - Define new data field flights_count as type flights_organizer=>counter

```
data         : flights_count  type flights_organizer=>counter
             .
```

   - Include the following statement prior to calling subroutine show_flights_count:

```
  flights_count              = flights_organizer=>singleton->get_flights_count( ).
```

6. In method get_test_flights_via_carrier, remove the describe table statement.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 6 test methods.

Remarks

With this version we have removed the attribute flights_count from class tester and instead rely on the instance of class flights_organizer to provide this value. It also undoes the change we made in the previous version to enable seeing that the unit test is using fabricated test data, meaning that again there is no unit test for subroutine show_flights.

Let’s register in our issues list that this version introduces new issue #29, with a reference to issue #26 that had been resolved already for this same reason.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue

#	Identified	Resolved	Description
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with

#	Identified	Resolved	Description
			ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F		Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	**ZAUT201L**		**Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)**

12.13 Exercise 71

Program: ZAUT201M

Requirements

Reason for change

- Provide unit test for method get_flights_count of class flights_organizer.

Changes to be applied

1. Add new test method get_flights_count to class tester:
   - Add method definition for get_flights_count to the private section of class tester after the definition for method get_flights_via_carrier:

```
methods      : o
               o
               o
             , get_flights_count
                 for testing
```

   - Include the following method implementation after the implementation for method get_flights_via_carrier:

```
method get_flights_count.
  clear flights_organizer=>singleton->flights_stack.
  cl_abap_unit_assert=>assert_equals(
    act                         = flights_organizer=>singleton->get_flights_count( )
    exp                         = 00
    msg                         = 'Flights stack is not initial'
    ).
  flights_organizer=>singleton->flights_stack
                                = test_flights_stack.
  cl_abap_unit_assert=>assert_equals(
    act                         = flights_organizer=>singleton->get_flights_count( )
    exp                         = lines( test_flights_stack )
    msg                         = 'Flights stack does not have expected number of entries'
    ).
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

Previous version ZAUT201E introduced new method get_flights_count for class flights_organizer. With this version we have provided a test for it with new unit test method get_flights_count of class tester.

13 ABAP Unit Testing 202 – Gaining Control Over Unit Test Coverage of Output

This section describes the requirements for the exercise programs associated with the Chapter 7 section titled Encapsulating Indirect Output Processes to Accommodate Unit Testing in the book Automated Unit Testing with ABAP.

13.1 Exercise 72

Program: ZAUT202A

Requirements

Reason for change

- Migrate flights report processing into a singleton class.

Changes to be applied

1. Define new singleton class flights_report after the definition for class flights_organizer:

```
class flights_report                  definition
                                      final
                                      create private
                                      .
  public section.
    class-data   : singleton      type ref
                                    to flights_report
                                      read-only
                 .
    class-methods: class_constructor
                 .
    methods      : show_flights
                     importing
                       alv_style_grid
                         type xflag
                     changing
                       flights_stack
                         type flights_organizer=>flights_list
                 .
  private section.
    methods      : set_alv_field_catalog
                     importing
                       structure_name
                         type tabname
                     changing
                       alv_fieldcat_stack
                         type slis_t_fieldcat_alv
                 , set_alv_function_module_name
                     importing
                       alv_style_grid
                         type xflag
                     changing
                       alv_display_function_module
                         type progname
                 .
endclass.
class flights_report                  implementation.
  method class_constructor.
    create object singleton.
  endmethod.
  method show_flights.
    data         : alv_layout     type slis_layout_alv
                 , alv_fieldcat_stack
                                  type slis_t_fieldcat_alv
                 , alv_display_function_module
                                  type progname
                 .
```

```
    " Set field catalog for presenting flights via ALV report:
    call method set_alv_field_catalog
      exporting
        structure_name            = flights_organizer=>flights_table_name
      changing
        alv_fieldcat_stack        = alv_fieldcat_stack
        .
    if alv_fieldcat_stack is initial.
      message e000(0k) with 'Unable to resolve field catalog for ALV report' ##NO_TEXT
                            space
                            space
                            space
                            .
    endif.
    " Set name of alv presentation function module based on user selection:
    call method set_alv_function_module_name
      exporting
        alv_style_grid            = alv_style_grid
      changing
        alv_display_function_module
                                  = alv_display_function_module
        .
    " Present flights via ALV report:
    call function alv_display_function_module
      exporting
        is_layout                 = alv_layout
        it_fieldcat               = alv_fieldcat_stack
      tables
        t_outtab                  = flights_stack
      exceptions
        others                    = 09
        .
    if sy-subrc ne 00.
      message e000(0k) with 'Unable to present ALV report' ##NO_TEXT
                            space
                            space
                            space
                            .
    endif.
  endmethod.
  method set_alv_field_catalog.
    " Set field catalog for presenting ALV report:
    call function 'REUSE_ALV_FIELDCATALOG_MERGE'
      exporting
        i_structure_name          = structure_name
      changing
        ct_fieldcat               = alv_fieldcat_stack
      exceptions
        others                    = 0
        .
  endmethod.
  method set_alv_function_module_name.
    constants    : alv_list_function_module
                                  type progname  value 'REUSE_ALV_LIST_DISPLAY'
                 , alv_grid_function_module
                                  type progname  value 'REUSE_ALV_GRID_DISPLAY'
                 .
    " Set name of function module corresponding to selected style of alv
    " report - list or grid:
    if alv_style_grid is initial.
      alv_display_function_module = alv_list_function_module.
    else.
      alv_display_function_module = alv_grid_function_module.
    endif.
  endmethod.
endclass.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we have created the second local singleton class for use by the production code. Its name is flights_report and it effectively encapsulates all program code relating to the presentation of the flights records. It is not yet being used, but simply is defined. Notice that all the unit tests still pass, confirming that its introduction into the program does not cause any of the existing unit tests to fail.

As we had noted with the previous singleton class defined for production, they come with their own baggage rendering them undesirable entities in some situations. In a subsequent exercise we will eliminate the singleton nature of this class.

Let's register in our issues list that this version introduces new issue #30.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle

#	Identified	Resolved	Description
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F		Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L		Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	**ZAUT202A**		**Second singleton class is introduced: flights_report**

13.2 Exercise 73

Program: ZAUT202B

Requirements

Reason for change

- Provide class tester access to private methods of class flights_report.

Changes to be applied

1. Define new empty interface flights_report_testable, placing it after interface flights_organizer_testable:

```
interface flights_report_testable.
endinterface.
```

2. On class flights_report include a friends clause naming interface flights_report_testable:

```
class flights_report                 definition
                                     o
                                     o
                                     friends flights_report_testable
                                     .
```

3. In class tester, include flights_report_testable on the interfaces statement, following interface flights_organizer_testable:

```
    interfaces   : o
                   o
                   o
                 , flights_report_testable
```

4. In class tester method set_alv_field_catalog, replace the call to subroutine set_alv_field_catalog with a call to private method set_alv_field_catalog of the singleton of class flights_report:

```
    call method flights_report=>singleton->set_alv_field_catalog
      exporting
        structure_name           = flights_organizer=>flights_table_name
      changing
        alv_fieldcat_stack       = alv_fieldcat_stack
        .
```

5. In class tester method set_alv_function_module_name, replace the calls to subroutine set_alv_function_module_name with calls to private method set_alv_function_module_name of the singleton of class flights_report:

```
    call method flights_report=>singleton->set_alv_function_module_name
      exporting
        alv_style_grid           = ...
      changing
        alv_display_function_module
                                 = alv_display_function_module
        .
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we have again enabled class tester to have public access to the private members of some other class. In this case it is to enable methods of class tester to invoke the private methods of class flights_report for the purpose of testing them.

Notice that we introduced new empty interface flights_report_testable and now class flights_report has a “friends” clause naming this interface, effectively offering friendship to any class implementing interface flights_report_testable. Class tester now implements interface flights_report_testable, so by association class tester has what amounts to public access to the private members of class flights_report.

13.3 Exercise 74

Program: ZAUT202C

Requirements

Reason for change

- Replace calls to subroutine show_flights with with call to method show_flights of class flights_report.

Changes to be applied

1. In subroutine present_report, replace statement ...

```
perform show_flights using via_grid
                  changing flights_stack.
```

... with statement:

```
call method flights_report=>singleton->show_flights
  exporting
    alv_style_grid           = via_grid
  changing
    flights_stack         = flights_stack
  .
```

2. In unit test method show_flights of class tester, replace statement ...

```
perform show_flights using abap_false
                  changing flights_organizer=>singleton->flights_stack.
```

... with statement:

```
call method flights_report=>singleton->show_flights
  exporting
    alv_style_grid          = abap_false
  changing
    flights_stack           = flights_organizer=>singleton->flights_stack
    .
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we have substituted calls to subroutine show_flights with calls to method show_flights of singleton class flights_report.

13.4 Exercise 75

Program: ZAUT202D

Requirements

Reason for change

- Remove unused subroutine show_flights.

Changes to be applied

1. Remove unused subroutine show_flights.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we have removed the unused subroutine show_flights.

13.5 Exercise 76

Program: ZAUT202E

Requirements

Reason for change

- Confirm class tester is using test data.

Changes to be applied

1. Uncomment the "for testing" clause of method show_flights of class tester.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears for flights of carrier 'LH'; Press back, exit, cancel or ESCape, then
ALV classic list appears for flights of carrier 'UA'; Press back, exit, cancel or ESCape, then
ALV classic list appears for flights of carrier 'AA'; Press back, exit, cancel or ESCape, then
Status message indicates Processed: 1 program, 1 test classes, 8 test methods.

Remarks

This version simply enables us to confirm that the test data created by class tester is being used with the unit tests.

Let’s register in our issues list that this version resolves issue #29.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights

#	Identified	Resolved	Description
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F		Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	**ZAUT201L**	**ZAUT202E**	**Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)**
30	ZAUT202A		Second singleton class is introduced: flights_report

13.6 Exercise 77

Program: ZAUT202F

Requirements

Reason for change

- Remove unused subroutines set_alv_field_catalog and set_alv_function_module_name.

Changes to be applied

1. Again comment out the "for testing" clause of method show_flights of class tester.
2. Delete subroutines set_alv_field_catalog and set_alv_function_module_name now that they no longer are called.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we have removed the unused subroutines set_alv_field_catalog and set_alv_function_module_name. We also have undone the change we made in the previous version to enable seeing that the unit test is using fabricated test data, meaning that again there is no unit test for subroutine show_flights.

Let’s register in our issues list that this version introduces new issue #31,with a reference to issue #29 that had been resolved already for this same reason..

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier

#	Identified	Resolved	Description
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier

#	Identified	Resolved	Description
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F		Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A		Second singleton class is introduced: flights_report
31	**ZAUT202F**		**Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)**

14 ABAP Unit Testing 301 – Introducing a Test Double for Input

This section describes the requirements for the exercise programs associated with the Chapter 8 section titled Using Test Doubles for Indirect Input in the book Automated Unit Testing with ABAP.

14.1 Exercise 78

Program: ZAUT301A

Requirements

Reason for change

- Introduce test double for class flights_organizer.

Changes to be applied

1. Define class flights_organizer_test_double, which is an exact copy of class flights_organizer with the name of the class changed as necessary. Place it immediately following class flights_organizer:

```
class flights_organizer_test_double    definition
                                       final
                                       create private
                                       friends flights_organizer_testable
                                       .
  public section.
    types        : flights_row    type sflight
                 , flights_list   type standard table
                                    of flights_row
                 , carrier        type s_carr_id
                 , counter        type int4
                 .
    constants    : flights_table_name
                                  type tabname   value 'SFLIGHT'
                 .
    class-data   : singleton      type ref
                                    to flights_organizer_test_double
                                         read-only
                 .
    data         : flights_stack  type flights_list
                                         read-only
                 .
    class-methods: class_constructor
                 .
    methods      : get_flights_via_carrier
                       importing
                         carrier
                           type carrier
                 , get_flights_count
                       returning
                         value(flights_count)
                           type counter
                 .
endclass.
class flights_organizer_test_double    implementation.
  method class_constructor.
    create object singleton.
  endmethod.
  method get_flights_via_carrier.
    clear flights_stack.
    if carrier is not initial.
      try.
        select *
          into table flights_stack
          from (flights_table_name)
         where carrid               eq carrier
              .
      catch cx_root ##NO_HANDLER ##CATCH_ALL.
        " Nothing to do other than intercept potential exception due to
```

```
        " invalid dynamic table name
      endtry.
    endif.
  endmethod.
  method get_flights_count.
    describe table flights_stack lines flights_count.
  endmethod.
endclass.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we’ve introduced our first test double into the program, one for handling input required by the program. The test double, class flights_organizer_test_double, is an exact copy of class test_organizer except for its name. Like class test_organizer it is a defined as a singleton class, and as we’ve noted before, singleton classes come with their own baggage rendering them undesirable entities in some situations. In a subsequent exercise we will eliminate the singleton nature of this class.

Let’s register in our issues list that this version introduces new issue #32.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name

#	Identified	Resolved	Description
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F		Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)

#	Identified	Resolved	Description
30	ZAUT202A		Second singleton class is introduced: flights_report
31	ZAUT202F		Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	**ZAUT301A**		**Third singleton class is introduced: flights_organizer_test_double**

14.2 Exercise 79

Program: ZAUT301B

Requirements

Reason for change

- Enable class flights_organizer_test double to create and use its own test data.

Changes to be applied

1. Change class flights_organizer_test double in the following ways:
   - In the definition portion define a private section to include:
     - Constants defining 3 airlines, copied from corresponding constants defined in class tester
     - Data field test_flights_stack of type flights_organizer_test_double=>flights_list
     - Definition for method constructor (no signature)

     When completed it should look like this:

```
private section.
  constants    : lufthansa      type s_carr_id value 'LH'
               , united_airlines
                                type s_carr_id value 'UA'
               , american_airlines
                                type s_carr_id value 'AA'
               .
  data         : test_flights_stack
                                type flights_organizer_test_double=>flights_list
               .
  methods      : constructor
               .
```

2. In the implementation portion provide an implementation for the following methods:
   - constructor, to contain a copy of the code from method class_setup of class tester.
   - get_flights_via_carrier, to contain a copy of the code from method get_test_flights_via_carrier of class tester.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we have changed the test double introduced in the previous version so that it creates its own test data instead of retrieving records from table SFLIGHT. This test data is generated by the constructor method using a copy of the code implemented for method class_setup of class tester to perform this same task.

14.3 Exercise 80

Program: ZAUT301C

Requirements

Reason for change

- Remove from class tester redundant ability to create its own test data.

Changes to be applied

1. Change class tester in the following ways:
   - Remove attribute test_flights_stack.
   - Remove definition and implementation of these methods:
     - class_setup
     - class_teardown
     - get_test_flights_via_carrier
   - In these methods:
     - setup
     - get_flights_via_carrier
     - show_flights

     replace this statement ...

```
call method get_test_flights_via_carrier
  exporting
    carrier                  = carrier
  changing
    flights_stack            = flights_organizer=>singleton->flights_stack
    .
```

… with this pair of statements:

```
call method flights_organizer_test_double=>singleton->get_flights_via_carrier
  exporting
    carrier                  = carrier
    .
flights_organizer=>singleton->flights_stack
                             = flights_organizer_test_double=>singleton->flights_stack.
```

   - In method get_flights_count, change references to attribute test_flights_stack to references to attribute flights_organizer_test_double=>singleton->test_flights_stack.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version, there no longer is a need for class tester also to have its own code to generate test data when it is a simple matter to call the methods of class flights_organizer_test_double to do it. Accordingly, class tester no longer needs its class_data attribute test_flights_stack, its static methods class_setup and class_teardown and its instance method get_test_flights_via_carrier, all of which are removed with this version.

14.4 Exercise 81

Program: ZAUT301D

Requirements

Reason for change

- Introduce into productive code the capability to run in unit test mode.

Changes to be applied

1. Define a selection screen parameter named unittest as a checkbox after the parameter via_grid.
2. In the "at selection-screen" classic event block, replace these statements ...

```
  " Get list of flights corresponding to specified carrier:
  call method flights_organizer=>singleton->get_flights_via_carrier
    exporting
      carrier                  = carrier.
  " Diagnose when no flights for this carrier:
  if flights_organizer=>singleton->get_flights_count( ) le 00.
    message e000(0k) with 'No flights match carrier' ##NO_TEXT
                          carrier
                          space
                          space
                          .
  endif.
```

… with these statements:

```
    " Get list of flights corresponding to specified carrier:
  if unittest is not initial.
    call method flights_organizer_test_double=>singleton->get_flights_via_carrier
      exporting
        carrier                  = carrier.
    " Diagnose when no flights for this carrier:
    if flights_organizer_test_double=>singleton->get_flights_count( ) le 00.
      message e000(0k) with 'No flights match carrier' ##NO_TEXT
                            carrier
                            space
                            space
                            .
    endif.
  else.
    call method flights_organizer=>singleton->get_flights_via_carrier
      exporting
```

```
      carrier                  = carrier.
    " Diagnose when no flights for this carrier:
    if flights_organizer=>singleton->get_flights_count( ) le 00.
      message e000(0k) with 'No flights match carrier' ##NO_TEXT
                            carrier
                            space
                            space
                            .
    endif.
  endif.
```

3. In subroutine present_report, check field unittest: when it is not blank:
   - Then set field flights_stack from flights_organizer_test_double=>singleton->flights_stack; otherwise set field flights_stack from flights_organizer=>singleton->flights_stack.
   - Then set field flights_count from call to flights_organizer_test_double=>singleton->get_flights_count( ); otherwise set field flights_count from call to flights_organizer=>singleton->get_flights_count( ).

   When completed the subroutine should look like this:

```
form present_report using discount
                          type discount
                        via_grid
                          type xflag
                        carrier
                          type flights_organizer=>carrier.
    data         : flights_stack  type flights_organizer=>flights_list
                 , flights_count  type flights_organizer=>counter
                 .
  if unittest is not initial.
    flights_stack                = flights_organizer_test_double=>singleton->flights_stack.
  else.
    flights_stack                = flights_organizer=>singleton->flights_stack.
  endif.
    " Adjust flights fare by specified discount:
    perform apply_flight_discount using discount
                                changing flights_stack.
    " Get total revenue for flight as currently booked:
    perform adjust_flight_revenue changing flights_stack.
  if unittest is not initial.
    flights_count                = flights_organizer_test_double=>singleton->get_flights_count( ).
  else.
    flights_count                = flights_organizer=>singleton->get_flights_count( ).
  endif.
    perform show_flights_count using flights_count
                                     carrier.
    call method flights_report=>singleton->show_flights
      exporting
        alv_style_grid           = via_grid
      changing
        flights_stack            = flights_stack
        .
endform.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list, unittest is unchecked and press Execute.

Result: Produces ALV classic list display using records from table sflight with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Action: Specify Airline ‘AA’, no discount, ALV classic list, unittest is checked and press Execute.

Result: Produces ALV classic list display of records using fabricated test records with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we took a short detour to show an example of how **not to** implement a unit test. Here we intentionally have provided a flag field on the selection screen with which the user of this program can designate whether the program should be run normally or in test mode. This means that the production code is sensitive to the fact that it can be run in test mode. **This is never a good idea.** It results in a program that behaves differently when run in test mode than when not. It represents an example of using a “Test Hook”, a unit testing pattern and one of the causes of the unit test smell known as "Test Logic in Production" cataloged by Gerard Meszaros (xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 217).

14.5 Exercise 82

Program: ZAUT301E

Requirements

Reason for change

- Begin the process of enabling classes flights_organizer and flights_organizer_test double to be accessible via the same interface reference variable.

Changes to be applied

1. Undo the changes made in the previous version (simply copy its preceding version).
2. Define new interface flights_organizable, placing it after interface flights_report_testable:

```
interface flights_organizable.
    types        : flights_row    type sflight
                 , flights_list   type standard table
                                    of flights_row
                 , carrier        type s_carr_id
                 , counter        type int4
                 .
    constants    : flights_table_name
                                  type tabname   value 'SFLIGHT'
                 .
    methods      : get_flights_via_carrier
                     importing
                       carrier
                         type carrier
                 , get_flights_count
                     returning
                       value(flights_count)
                         type counter
                 .
endinterface.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we have removed the abominable implementation introduced in the previous version to allow the productive code to run in either normal mode or in test mode. In addition we've introduced a local interface called flights_organizable. Defining such an interface is the first step toward providing the capability for both classes flights_organizer and flights_organizer_test_double to become interchangeable entities. Notice that this interface provides the same types, constants and methods definitions found in both classes flights_organizer and flights_organizer_test_double.

14.6 Exercise 83

Program: ZAUT301F

Requirements

Reason for change

- Continue the process of enabling classes flights_organizer and flights_organizer_test double to be accessible.

Changes to be applied

1. Change public section of class flights_organizer as follows:
   - Include an interfaces statement referencing interface flights_organizable, after the public section header.
   - Include an aliases statement defining aliases for the methods contributed by interface flights_organizable, after the interfaces statement.
   - Remove the following:
     - types statements
     - constants statements
     - methods statements
   - Change attribute flights_stack from type flights_list to type flights_organizable=>flights_list.

   Afterward, the public section of class flights_organizer should look like this:

```
public section.
  interfaces   : flights_organizable
               .
  aliases      : get_flights_via_carrier
                   for flights_organizable~get_flights_via_carrier
               , get_flights_count
                   for flights_organizable~get_flights_count
               .
  class-data   : singleton      type ref
                                  to flights_organizer
                                       read-only
               .
  data         : flights_stack  type flights_organizable=>flights_list
                                       read-only
               .
  class-methods: class_constructor
               .
```

2. Change references to former attributes of flights_organizer to now reference their counterparts in interface flights_organizable.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we have removed from class flights_organizer everything that it acquires by implementing interface flights_organizable. Notice that the definition of its singleton field has changed from type ref to class flights_organizer to type ref to interface flights_organizable.

This also has a ripple effect on class flights_report and the selection screen field carrier since they referred to entities no longer defined in class flights_organizer but that now exist in interface flights_organizable.

14.7 Exercise 84

Program: ZAUT301G

Requirements

Reason for change

- Continue the process of enabling classes flights_organizer and flights_organizer_test double to be accessible via the same interface reference variable.

Changes to be applied

1. Change public section of class flights_organizer_test_double as follows:
   - Include an interfaces statement referencing interface flights_organizable, after the public section header.
   - Include an aliases statement defining aliases for the methods contributed by interface flights_organizable, after the interfaces statement.
   - Remove the following:
     - types statements
     - constants statements
     - methods statements
   - Change attribute flights_stack from type flights_list to type flights_organizable=>flights_list.

   Afterward, the public section of class flights_organizer should look like this:

```
public section.
  interfaces   : flights_organizable
               .
  aliases      : get_flights_via_carrier
                   for flights_organizable~get_flights_via_carrier
               , get_flights_count
                   for flights_organizable~get_flights_count
               .
  class-data   : singleton      type ref
                                  to flights_organizer_test_double
                                       read-only
```

```
             .
data         : flights_stack  type flights_organizable=>flights_list
                                   read-only
             .
class-methods: class_constructor
             .
```

2. In private section, change reference to former attributes of flights_organizer_test_double to now reference their counterparts in interface flights_organizable.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we have done for class flights_organizer_test_double the same that was done in the previous version for class flights_organizer. Now both classes flights_organizer and flights_organizer_test_double implement interface flights_organizable. This means that a variable defined as reference to interface flights_organizable may hold a reference to an instance of either of these classes.

14.8 Exercise 85

Program: ZAUT301H

Requirements

Reason for change

- Complete the process of enabling classes flights_organizer and flights_organizer_test double to be accessible via the same interface reference variable.

Changes to be applied

1. In interface flights_organizable:
   - Copy data field flights_stack of class flights_organizer, dropping the read-only qualifier, placing it after the constants statement.
2. In class flights_organizer:
   - Provide alias flights_stack for flights_organizable~flights_stack after the one for get_flights_count.
   - Change singleton definition to type ref to flights_organizable.
   - Remove data field flights_stack.
   - Change class_constructor method of class flights_organizer to qualify the create object statement with "type flights_organizer".
3. In class flights_organizer_test_double:
   - Provide alias flights_stack for flights_organizable~flights_stack after the one for get_flights_count.
   - Change singleton definition to type ref to flights_organizable.

- Remove data field flights_stack.
- Change class_constructor method of class flights_organizer to qualify the create object statement with "type flights_organizer_test_double".

4. In method get_flights_count of class tester:
   - After the method statement define field flights_organizable as ref to flights_organizer_test_double:

```
data        : flights_organizable
                    type ref
                      to flights_organizer_test_double
            .
```

   - Immediately prior to the statement setting the value of flights_organizer=>singleton->flights_stack from flights_organizer_test_double->test_flights_stack, insert try-catch-endtry block for moving the value of reference to flights_organizer_test_double=>singleton to flights_organizable, catching exception cx_sy_move_cast_error with an appropriate aunit_assert=>fail message:

```
try.
  flights_organizable         ?= flights_organizer_test_double=>singleton.
catch cx_sy_move_cast_error.
  cl_abap_unit_assert=>fail(
    msg                       = 'Caught exception in test method get_flights_count'
    ).
endtry.
```

   - Change references to flights_organizer_test_double=>singleton->test_flights_stack to reference instead flights_organizable->test_flights_stack.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we’ve extracted attribute flights_stack from both classes flights_organizer and flights_organizer_test_double and moved it into interface flights_organizable. Both classes now provide an alias to continue to refer to this attribute by the name flights_stack, now that it is provided to both classes through an interface.

Note: A side effect of relocating attribute flights_stack from both classes to the interface is that now it no longer carries the “read-only” qualifier. Had we retained the “read-only” qualifier with the interface definition, it would have rendered the syntax invalid for the methods of class tester which have statements changing this attribute, a total of 9 statements across 6 methods.

My expectation had been that because class tester has friendship with both of the classes that implement interface flights_organizable that it would continue to have change access to the members these classes gain through any interface they had specified on an interfaces statement, the same change access offered by these classes when the attribute was declared in the classes themselves. Apparently that is not the case. Accordingly, attribute flights_stack for both classes flights_organizer and flights_organizer_test_double is now a public attribute that can be changed by any external entity,

an undesirable exposure that we will register in our issues list.

A potential resolution for this is to retain the “read-only” qualifier with the attribute defined in the interface and also to define an accompanying public method for the interface, to be implemented by the classes, which external entities can call to request the flights_stack attribute be updated for them. The problem with this approach is that changing the content of the flights_stack attribute is required only by the unit test methods of this program and not by any of its production code, so defining such a method to the interface would be to provide a capability to the implementing classes that the production code would not use. It would represent another example of the production code being aware that it can be tested, the code existing “For Tests Only”, a unit testing pattern and one of the causes of the unit test smell known as "Test Logic in Production" (xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 217).

Notice that now both classes flights_organizer and flights_organizer_test_double have their respective singleton attributes defined as a reference to interface flights_organizable instead of to their respective class names, and also have included a “type” clause on the create object statement in their respective class_constructor methods to indicate which type of instance to create.

Notice also that now unit test method get_flights_count of class tester requires a specializing cast to move the instance bound to the singleton attribute of class flights_organizer_test_double to a reference variable specifically defined as a reference to class flights_organizer_test_double. This maneuver, surrounded by a try-endtry block, is required in order to gain access to attribute test_flights_stack of class flights_organizer_test_double, an attribute defined neither in class flights_organizer nor in interface flights_organizable. Once the unit test determines that the instance bound to the singleton attribute of class flights_organizer_test_double is indeed an instance of class flights_organizer_test_double, it can then use the number of rows in attribute test_flights_stack as the expected count of flights that should satisfy the assertion.

Let’s register in our issues list that this version introduces new issue #33.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name

#	Identified	Resolved	Description
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F		Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)

#	Identified	Resolved	Description
30	ZAUT202A		Second singleton class is introduced: flights_report
31	ZAUT202F		Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A		Third singleton class is introduced: flights_organizer_test_double
33	**ZAUT301H**		**Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship**

14.9 Exercise 86

Program: ZAUT301I

Requirements

Reason for change

- Substitute instance of class flights_organizer with instance of class flights_organizer_test_double during unit test.

Changes to be applied

1. In class tester method setup:
   - Insert ahead of first statement:

```
" Override the singleton in class flights_organizer with the singleton from
" class flights_organizer_test_double, effectively causing all references
" to flights_organizer=>singleton to be redirected to referencing the same
" singleton created by the class_constructor of flights_organizer_test_double:
flights_organizer=>singleton  = flights_organizer_test_double=>singleton.
```

   - Change call to method flights_organizer_test_double=>singleton->get_flights_via_carrier to instead call method flights_organizer=>singleton->get_flights_via_carrier.
   - Discard the statement setting flights_organizer=>singleton->flights_stack.
2. In class tester method get_flights_via_carrier:
   - Change call to method flights_organizer_test_double=>singleton->get_flights_via_carrier to instead call method flights_organizer=>singleton->get_flights_via_carrier.
   - Discard the statement setting flights_organizer=>singleton->flights_stack.
3. In class tester method get_flights_count:
   - Set value of flights_organizable from flights_organizer=>singleton instead of from flights_organizer_test_double=>singleton.
4. In class tester method show_flights:
   - Change call to method flights_organizer_test_double=>singleton->get_flights_via_carrier to instead call method flights_organizer=>singleton->get_flights_via_carrier.
   - Discard the statement setting flights_organizer=>singleton->flights_stack.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 7 test methods.

Remarks

With this version we now have the production code as well as the unit test code of class tester both referring to flights_organizer=>singleton for all their flights information. However, because we have included code in the setup method of class tester to replace the value of flights_organizer=>singleton with the reference to flights_organizer_test_double=>singleton, it means the unit tests are actually using the instance of flights_organizer_test_double for all their calls to the instance bound to flights_organizer=>singleton.

This became possible with the previous version when we changed the definitions of the singleton static fields in both flights_organizer and flights_organizer_test_double from references to their own class names, respectively, to both now having their singleton static fields defined as references to interface flights_organizable. This is why field flights_organizer=>singleton can hold a reference to an instance of flights_organizer_test_double – because it is now defined as a reference to interface flights_organizable, and class flights_organizer_test_double implements that interface.

Notice that now unit tests get_flights_via_carrier, get_flights_count and show_flights have been changed to no longer reference the instance bound to flights_organizer_test_double=>singleton, but instead now reference the instance bound to flights_organizer=>singleton.

Let's register in our issues list that this version affects issue #28 because we are now using the instance bound to the singleton attribute of flights_organizer when running the unit tests.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A		No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name

#	Identified	Resolved	Description
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	**ZAUT201F**	**(ZAUT301I)**	**Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)**
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with

#	Identified	Resolved	Description
			ZAUT201K; see issue #26)
30	ZAUT202A		Second singleton class is introduced: flights_report
31	ZAUT202F		Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A		Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship

15 ABAP Unit Testing 302 – Introducing a Test Double for Output

This section describes the requirements for the exercise programs associated with the Chapter 8 section titled Using Test Doubles for Indirect Output in the book Automated Unit Testing with ABAP.

15.1 Exercise 87

Program: ZAUT302A

We have identified the following issues during our progress to this point:

1. Certain types of output interfere with a clean test run:
   a) A MESSAGE statement with severity error, abort or exit will result in test failures.
   b) ALV output produced during a test will require user intervention to allow the test to continue.
   c) Certain list commands, such as a WRITE statement, will produce the Internal Session for Isolated Test Class Execution screen, not only diagnosing the use of such statements but also requiring user intervention to allow the test to continue.
2. There remains no test for subroutine present_report.

Requirements

Reason for change

- Implement unit test for subroutine present_report (the only remaining subroutine without a unit test).

Changes to be applied

1. In class tester, create new test method present_report:
   - Add method definition for present_report to the private section of class tester after the definition for method show_flights_count:

```
  methods      : o
                 o
                 o
               , present_report
                   for testing
```

   - Include the following method implementation after the implementation for method adjust_flight_revenue:

```
method present_report.
  constants    : no_discount    type discount  value 00
               , alv_classic_list
                              type abap_bool value abap_false
               .
  set_bogus_message( ).
  assert_message_is_bogus( ).
  perform present_report using no_discount
                               alv_classic_list
                               carrier.
  assert_message_not_bogus( ).
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears showing test data records for flights of carrier 'AA'; Press back, exit, cancel or ESCape, then returns to editor screen;
Status message indicates Processed: 1 program, 1 test classes, 8 test methods.

Remarks

With this version we have defined new unit test method present_report to class tester. It asserts that a message is issued after calling subroutine present_report.

Similar to the unit testing results we saw when we enabled the “for testing” clause for unit test show_flights of class tester, here we see that having a unit test which calls subroutine present_report, which until this version was the only remaining subroutine without a unit test, also will produce an ALV list, requiring user interaction to allow the unit test to continue to completion.

Let’s register in our issues list that this version resolves issue #7.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	**ZAUT101A**	**ZAUT302A**	**No test for code in subroutine present_report**
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module

#	Identified	Resolved	Description
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A		Second singleton class is introduced: flights_report
31	ZAUT202F		Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A		Third singleton class is introduced: flights_organizer_test_double

#	Identified	Resolved	Description
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship

15.2 Exercise 88

Program: ZAUT302B

Requirements

Reason for change

- Introduce test double for class flights_report.

Changes to be applied

1. Define class flights_report_test_double, which is a copy of class flights_report with the name of the class changed as necessary and the implementation code for method show_flights removed. Place it immediately following class flights_report:

```
class flights_report_test_double       definition
                                       final
                                       create private
                                       friends flights_report_testable
                                       .
  public section.
    class-data   : singleton      type ref
                                    to flights_report_test_double
                                         read-only
                 .
    class-methods: class_constructor
                 .
    methods      : show_flights
                       importing
                         alv_style_grid
                           type xflag
                       changing
                         flights_stack
                           type flights_organizable=>flights_list
                 .
  private section.
    methods      : set_alv_field_catalog
                       importing
                         structure_name
                           type tabname
                       changing
                         alv_fieldcat_stack
                           type slis_t_fieldcat_alv
                 , set_alv_function_module_name
                       importing
                         alv_style_grid
                           type xflag
                       changing
                         alv_display_function_module
                           type progname
                 .
endclass.
class flights_report_test_double       implementation.
  method class_constructor.
    create object singleton.
  endmethod.
  method show_flights.
  endmethod.
  method set_alv_field_catalog.
    " Set field catalog for presenting ALV report:
```

```
    call function 'REUSE_ALV_FIELDCATALOG_MERGE'
      exporting
        i_structure_name          = structure_name
      changing
        ct_fieldcat               = alv_fieldcat_stack
      exceptions
        others                    = 0
        .
  endmethod.
  method set_alv_function_module_name.
    constants    : alv_list_function_module
                                  type progname  value 'REUSE_ALV_LIST_DISPLAY'
                 , alv_grid_function_module
                                  type progname  value 'REUSE_ALV_GRID_DISPLAY'
                 .
    " Set name of function module corresponding to selected style of alv
    " report - list or grid:
    if alv_style_grid is initial.
      alv_display_function_module = alv_list_function_module.
    else.
      alv_display_function_module = alv_grid_function_module.
    endif.
  endmethod.
endclass.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears showing test data records for flights of carrier 'AA'; Press back, exit, cancel or ESCape, then returns to editor screen;
Status message indicates Processed: 1 program, 1 test classes, 8 test methods.

Remarks

With this version we’ve introduced another test double, this time for handling output produced by the program. The test double, class flights_report_test_double, is an exact copy of class test_report except for its name and that its implementation for method show_flights is empty. Like class test_organizer it is defined as a singleton class, and as we’ve noted before, singleton classes come with their own baggage rendering them undesirable entities in some situations. In a subsequent exercise we will eliminate the singleton nature of this class.

Let’s register in our issues list that this version introduces new issue #34.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare

#	Identified	Resolved	Description
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)

#	Identified	Resolved	Description
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A		Second singleton class is introduced: flights_report
31	ZAUT202F		Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A		Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	**ZAUT302B**		**Fourth singleton class is introduced: flights_report_test_double**

15.3 Exercise 89

Program: ZAUT302C

Requirements

Reason for change

- Activate unit test method show_flights of class tester (deactivated since 202F).

Changes to be applied

1. In class tester, uncomment the "for testing" clause of method show_flights.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears showing test data records for flights of carrier 'AA'; Press back, exit, cancel or ESCape, then
ALV classic list appears showing test data records for flights of carrier 'LH'; Press back, exit, cancel or ESCape, then
ALV classic list appears showing test data records for flights of carrier 'UA'; Press back, exit, cancel or ESCape, then
ALV classic list appears showing test data records for flights of carrier 'AA'; Press back, exit, cancel or ESCape, then returns to editor screen;
Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have reactivated the “for testing” clause of unit test method show_flights of class tester, enabling us once again to see the ALV reports produced when method show_flights calls subroutine show_flights.

Let’s register in our issues list that this version resolves issue #31.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle

#	Identified	Resolved	Description
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L		ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A		Second singleton class is introduced: flights_report
31	**ZAUT202F**	**ZAUT302C**	**Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)**
32	ZAUT301A		Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B		Fourth singleton class is introduced: flights_report_test_double

#	Identified	Resolved	Description

15.4 Exercise 90

Program: ZAUT302D

Requirements

Reason for change

- Suppress flights report produced by method show_flights of class tester during unit test run.

Changes to be applied

1. Change method show_flights of class tester to call method show_flights of class flights_report_test_double instead of class flights_report.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears showing test data records for flights of carrier 'AA'; Press back, exit, cancel or ESCape, then returns to editor screen;
Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have changed the unit test method show_flights so that instead of calling method show_flights of singleton class flights_report it calls method show_flights of class flights_report_test_double. Since this is an empty method, the ALV reports no longer appear during the running of the unit tests, and now require no user intervention to allow the unit tests to run to completion. The fact that unit test method show_flights explicitly calls a test double is problematic also, and we will address this in a subsequent exercise program.

Let’s register in our issues list that this version resolves issue #22, but also introduces new issue #35.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount

#	Identified	Resolved	Description
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	**ZAUT105L**	**ZAUT302D**	**ALV report presented during unit test requires user intervention to continue test to completion**
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L;

#	Identified	Resolved	Description
			see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A		Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A		Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B		Fourth singleton class is introduced: flights_report_test_double
35	**ZAUT302D**		**Unit test method show_flights of class tester explicitly calls test double**

15.5 Exercise 91

Program: ZAUT302E

Requirements

Reason for change

- Enable class flights_report_test_double to record the number of times its sole public method is called.

Changes to be applied

1. In class flights_report_test_double, define private attribute number_of_calls type int4 following private section header:

```
data        : number_of_calls
                        type int4
            .
```

2. Change method show_flights in class tester to perform an assertion on this attribute upon exiting the loop through all the carrier ids:

```
cl_abap_unit_assert=>assert_equals(
  act                       = flights_report_test_double=>singleton->number_of_calls
  exp                       = lines( carrier_id_stack )
  msg                       = 'Unexpected number of calls to method'
  ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears showing test data records for flights of carrier 'AA'; Press back, exit, cancel or ESCape, then
ABAP Unit: Results Display report indicates test method show_flights triggers failure; Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have introduced a new attribute into class flights_report_test_double to keep track of the number of calls made to method show_flights, and in unit test show_flights of class tester have included an assertion that this number of calls is equal to the number of carriers for which flights are to be reported. The unit test method show_flights now fails because we have not provided any implementation in method show_flights of singleton class flights_report to keep track of the number of times is was called.

15.6 Exercise 92

Program: ZAUT302F

Requirements

Reason for change

- Fix the unit test failures encountered in the previous version.

Changes to be applied

1. Change method show_flights of class flights_report_test_double to increment attribute number_of_calls each time the method is invoked:

```
method show_flights.
  add 01 to number_of_calls.
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears showing test data records for flights of carrier 'AA'; Press back, exit, cancel or ESCape, then returns to editor screen;
Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have provided an implementation for method show_flights of singleton class flights_report_test_double to increment the counter number_of_calls each time a call is made to method show_flights.

Once again, all unit tests pass.

Class flights_report_test_double now becomes what Gerard Meszaros refers to as a test spy – a test double that is capable of recording information for later reference about how it has been used during the unit test (see xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 137). With this test spy we are able to call it to perform the "Behavior Verification" (xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 468) confirming that method show_flights of class flights_report_test_double was called the number of times we expect it should have been called.

15.7 Exercise 93

Program: ZAUT302G

Requirements

Reason for change

- Begin the process of enabling classes flights_report and flights_report_test double to be accessible via the same interface reference variable.

Changes to be applied

1. Define new interface flights_reportable, following interface flights_organizable, to contain the following method definition from the public section of class flights_report:

```
interface flights_reportable.
    methods      : show_flights
                       importing
                         alv_style_grid
                           type xflag
                       changing
                         flights_stack
                           type flights_organizable=>flights_list
                   .
endinterface.
```

Run

Action: Specify Airline 'AA', no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears showing test data records for flights of carrier 'AA'; Press back, exit, cancel or ESCape, then returns to editor screen;
Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have introduced another local interface called flights_reportable. Defining such an interface is the first step toward providing the capability for both classes flights_report and flights_report_test_double to become interchangeable entities. Notice that this interface provides the same definition for method show_flights as also found in both classes flights_report and flights_report_test_double.

15.8 Exercise 94

Program: ZAUT302H

Requirements

Reason for change

- Continue the process of enabling classes flights_report and flights_report_test double to be accessible via the same interface reference variable.

Changes to be applied

1. In class flights_report:
   - Replace the public section with the following:

```
public section.
  interfaces   : flights_reportable
                 .
  aliases      : show_flights
                   for flights_reportable~show_flights
                 .
  class-data   : singleton      type ref
                                  to flights_reportable
                                       read-only
                 .
  class-methods: class_constructor
                 .
```

   - Change method class_constructor to qualify the create object statement with "type flights_report":

```
method class_constructor.
  create object singleton type flights_report.
endmethod.
```

2. In method set_alv_field_catalog of class tester:
   - Define new data field: flights_report type ref to flights_report:

```
  data         : o
                 o
```

```
              o
            , flights_reportable
                           type ref
                             to flights_report
```

- Immediately prior to the statement calling method set_alv_field_catalog of class flights_report, insert try-catch-endtry block for moving the value of reference to flights_report=>singleton to flights_report, catching exception cx_sy_move_cast_error with an appropriate aunit_assert=>fail message:

```
try.
  flights_reportable          ?= flights_report=>singleton.
catch cx_sy_move_cast_error.
  cl_abap_unit_assert=>fail(
    msg                       = 'Caught exception in test method set_alv_field_catalog'
    ).
endtry.
```

- Change call to method flights_report=>singleton->set_alv_field_catalog to instead call method flights_reportable->set_alv_field_catalog.

3. In method set_alv_function_module_name of class tester:
   - Define new data field: flights_reportable type ref to flights_report:

```
data        : o
              o
              o
            , flights_reportable
                           type ref
                             to flights_report
```

- Immediately prior to the statement calling method set_alv_function_module_name of class flights_report, insert try-catch-endtry block for moving the value of reference to flights_report=>singleton to flights_reportable, catching exception cx_sy_move_cast_error with an appropriate aunit_assert=>fail message:

```
try.
  flights_reportable          ?= flights_report=>singleton.
catch cx_sy_move_cast_error.
  cl_abap_unit_assert=>fail(
    msg                       = 'Caught exception in test method set_alv_function_module_name'
    ).
endtry.
```

- Change both calls to method flights_report=>singleton->set_alv_function_module_name to instead call method flights_reportable->set_alv_function_module_name.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears showing test data records for flights of carrier 'AA'; Press back, exit, cancel or ESCape, then returns to editor screen;
Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have removed from class flights_report everything that it acquires by implementing interface flights_reportable. Notice that the definition of its singleton field has changed from type ref to class flights_report to type ref to interface flights_reportable.

This also has a ripple effect on methods set_alv_field_catalog and set_alv_function_module_name of class tester because these unit test methods are invoking methods that are not available via a reference to interface flights_reportable. Accordingly, each of these unit test methods moves the reference found in field flights_report=>singleton into a field defined explicitly as a reference to class flights_report, then makes the associated method call using that reference.

15.9 Exercise 95

Program: ZAUT302I

Requirements

Reason for change

- Complete the process of enabling classes flights_report and flights_report_test double to be accessible via the same interface reference variable.

Changes to be applied

1. In class flights_report_test_double:
   - Replace the public section with the following:

```
public section.
  interfaces   : flights_reportable
                 .
  aliases      : show_flights
                   for flights_reportable~show_flights
                 .
  class-data   : singleton      type ref
                                  to flights_reportable
                                       read-only
                 .
  class-methods: class_constructor
                 .
```

   - Change method class_constructor to qualify the create object statement with "type flights_report_test_double":

```
method class_constructor.
  create object singleton type flights_report_test_double.
endmethod.
```

2. In method show_flights of class tester:
   - Define new data field: flights_report_test_double type ref to flights_report_test_double:

```
  data         : o
                 o
                 o
               , flights_reportable
                             type ref
                               to flights_report_test_double
```

   - Immediately after the endloop statement, insert try-catch-endtry block for moving the value of reference to flights_report_test_double=>singleton to flights_reportable, catching exception cx_sy_move_cast_error with an appropriate aunit_assert=>fail message:

```
try.
  flights_reportable          ?= flights_report_test_double=>singleton.
catch cx_sy_move_cast_error.
  cl_abap_unit_assert=>fail(
    msg                       = 'Caught exception in test method show_flights'
    ).
endtry.
```

- On the call to static method cl_abap_unit_assert=>assert_equals, change the act parameter from reference to flights_report_test_double=>singleton->number_of_calls to instead reference to flights_reportable->number_of_calls.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ALV classic list appears showing test data records for flights of carrier 'AA'; Press back, exit, cancel or ESCape, then returns to editor screen;
Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have done for class flights_report_test_double the same that was done in the previous version for class flights_report. Now both classes flights_report and flights_report_test_double implement interface flights_reportable. This means that a variable defined as reference to interface flights_reportable may hold a reference to an instance of either of these classes.

This also has a ripple effect on method show_flights of class tester because it is invoking a method that is not available via a reference to interface flights_reportable. Accordingly, this unit test method moves the reference found in field flights_report_test_double=>singleton into a field defined explicitly as a reference to class flights_report_test_double, then makes the associated method call using that reference.

15.10 Exercise 96

Program: ZAUT302J

Requirements

Reason for change

- Substitute instance of class flights_report with instance of class flights_report_test_double during unit test.

Changes to be applied

1. In method setup of class tester prior to the statement setting field carrier, override flights_report=>singleton with flights_report_test_double=>singleton:

```
" Override the singleton in class flights_report with the singleton from
```

```
" class flights_report_test_double, effectively causing all references
" to flights_report=>singleton to be redirected to referencing the same
" singleton created by the class_constructor of flights_report_test_double:
flights_report=>singleton     = flights_report_test_double=>singleton.
```

Run

Action: Specify Airline 'AA', no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers failures for methods set_alv_field_catalog, set_alv_function_module_name and show_flights;
Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we now have the production code as well as the unit test code of class tester both referring to flights_report=>singleton for all their reporting. However, because we have included code in the setup method of class tester to replace the value of flights_report=>singleton with the reference to flights_report_test_double=>singleton, it means the unit tests are actually using the instance of flights_report_test_double for all their calls to flights_report=>singleton.

This became possible with the previous version when we changed the definitions of the singleton static fields in both flights_report and flights_report_test_double from references to their own class names, respectively, now to having both their singleton static fields defined as references to interface flights_reportable. This is why field flights_report=>singleton can hold a reference to an instance of flights_report_test_double – because it is now defined as a reference to interface flights_reportable, and class flights_report_test_double implements that interface.

The assertion in unit test method show_flights of class tester fails because singleton class flights_report_test_double, masquerading as singleton class flights_report, has its method show_flights called once during unit test method present_report and three more times in the loop found in unit test method show_flights, for a total of 4 times. The assertion in method show_flights expects the method to be called only 3 times, once for each of the carriers it uses for testing.

This illustrates one of the problems associated with using singleton classes: once they are established they remain in their last used state until the program ends. In this case, the attribute number_of_calls is set to 1 by method show_flights of class flights_report_test_double after being called via unit test method present_report, and it remains set to 1 when unit test method show_flights begins to execute and calls it 3 more times. In effect, there was no clearing of the number_of_calls attribute of the singleton class prior to the unit test method show_flights starting execution.

This illustrates unit tests exuding the unit test smell "Interacting Tests" cataloged by Gerard Meszaros (see xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 455). We cannot predict the order in which unit tests are executed, but the results from this example suggests that unit test present_report was run before unit test show_flights. The results of calling method show_flights of singleton class flights_report_test_double by unit test method present_report are left behind to interact with subsequent calls to method show_flights of singleton class flights_report_test_double by unit test method show_flights. Accordingly, unit test show_flights is not starting with a clean slate.

Let’s register in our issues list that this version introduces new issue #36.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with

#	Identified	Resolved	Description
			severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A		Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A		Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B		Fourth singleton class is introduced: flights_report_test_double
35	ZAUT302D		Unit test method show_flights of class tester explicitly calls test double
36	**ZAUT302J**		**Use of singleton classes result in interacting tests**

15.11 Exercise 97

Program: ZAUT302K

Requirements

Reason for change

- Change method present_report of class tester to assert the number of times a call was made to the instance of flights_reportable.

Changes to be applied

1. In method present_report of class tester:
   - Define field flights_reportable as ref to flights_report_test_double, placing it after the constants statement:

```
data        : flights_reportable
                        type ref
                          to flights_report_test_double
            .
```

   - After the call to method assert_message_not_bogus, insert try-catch-endtry block for moving the value of reference to flights_report=>singleton to flights_reportabe, catching exception cx_sy_move_cast_error with an appropriate aunit_assert=>fail message:

```
try.
  flights_reportable          ?= flights_report=>singleton.
catch cx_sy_move_cast_error.
  cl_abap_unit_assert=>fail(
    msg                       = 'Caught exception in test method present_report'
    ).
endtry.
```

   - After the new endtry statement, insert a test assertion that that the value of flights_reportable->number_of_calls is 01:

```
cl_abap_unit_assert=>assert_equals(
  act                         = flights_reportable->number_of_calls
  exp                         = 01
  msg                         = 'Unexpected number of calls to method'
  ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report still indicates test class still triggers failures for methods set_alv_field_catalog, set_alv_function_module_name and show_flights;
Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have included an assertion in unit test method present_report that only a single call had been made to singleton class flights_report_test_double. The same unit test failures we saw before appear again, but unit test method present_report still passes even with the additional assertion.

15.12 Exercise 98

Program: ZAUT302L

Requirements

Reason for change

- Reduce number of unit test methods triggering failures.

Changes to be applied

1. In method set_set_alv_field_catalog of class tester:
   - Change field flights_reportable from ref to flights_report to ref to flights_report_test_double.
2. In method set_alv_function_module_name of class tester:
   - Change field flights_reportable from ref to flights_report to ref to flights_report_test_double.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class triggers failure for method show_flights; Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have resolved the failures encountered with unit tests set_alv_field_catalog and set_alv_function_module_name. The failure of unit test show_flights still remains.

15.13 Exercise 99

Program: ZAUT302M

Requirements

Reason for change

- Eliminate remaining unit test failure.

Changes to be applied

1. At the top of method setup of class tester, create new objects for all 4 singletons:

```
" To prevent the possibility of interacting tests, refresh all singleton objects:
create object flights_organizer=>singleton
         type flights_organizer.
create object flights_organizer_test_double=>singleton
         type flights_organizer_test_double.
```

```
create object flights_report=>singleton
         type flights_report.
create object flights_report_test_double=>singleton
         type flights_report_test_double.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we’ve changed the setup method of class tester to create new instances of singleton classes into their respective singleton attributes, effectively causing each singleton class to be created anew and starting with a clean slate, and resolving the unit test failure caused by interacting tests. Now all the unit tests pass.

Let’s register in our issues list that this version resolves issue #36.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count

#	Identified	Resolved	Description
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A		Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)

#	Identified	Resolved	Description
32	ZAUT301A		Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B		Fourth singleton class is introduced: flights_report_test_double
35	ZAUT302D		Unit test method show_flights of class tester explicitly calls test double
36	**ZAUT302J**	**ZAUT302M**	**Use of singleton classes result in interacting tests**

15.14 Exercise 100

Program: ZAUT302N

Requirements

Reason for change

- Remove from method show_flights of class tester the explicit references to a test double.

Changes to be applied

1. In method show_flights of class tester:
   - Change references to flights_report_test_double=>singleton to references to flights_report=>singleton.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have resolved the issue where unit test method show_flights of class tester was invoking an explicit test double – flights_report_test_double. Now it is using the singleton attribute of class flights_report.

Let’s register in our issues list that this version resolves issue #35.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X

#	Identified	Resolved	Description
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A		Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A		Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B		Fourth singleton class is introduced: flights_report_test_double
35	**ZAUT302D**	**ZAUT302N**	**Unit test method show_flights of class tester explicitly calls test double**
36	ZAUT302J	ZAUT302M	Use of singleton classes result in interacting tests

16 ABAP Unit Testing 401 – Introducing a Service Locator

This section describes the requirements for the exercise programs associated with the Chapter 9 section titled Using a Service Locator in the book Automated Unit Testing with ABAP.

16.1 Exercise 101

Program: ZAUT401A

Requirements

Reason for change

- Introduce service locator capability for managing access to flights_organizer service.

Changes to be applied

1. Define new class service_locator ahead of class flights_organizer:

```
class service_locator                 definition
                                      final
                                      create private
                                      .
  public section.
    class-data   : singleton      type ref
                                    to service_locator
                                         read-only
                 .
    data         : flights_organizer
                                  type ref
                                    to flights_organizable
                 .
    class-methods: class_constructor
                 .
endclass.
class service_locator                 implementation.
  method class_constructor.
    create object service_locator=>singleton.
  endmethod.
endclass.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have introduced a new local class: service_locator. With it we are setting the stage for using a service locator to provide the services required by the program.

The term “service locator” is the name of a software design pattern enabling the control and replacement of dependencies during program execution. Among others of its design pattern synonyms is “Dependency Lookup”, the name by which Gerard Meszaros describes this design pattern. Here is his explanation for the capability a service locator provides for automated unit testing (xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 686):

> “Almost every piece of code depends on some other classes, objects, modules, or procedures. To unit-test a piece of code properly, we would like to isolate it from its dependencies. Such isolation is difficult to achieve, however, if those dependencies are hard-coded within the code in the form of literal [component names]. *Dependency Lookup* is a way to allow the normal coupling between a [component under test] and its dependencies to be broken during automated testing.
>
> How It Works
>
> We avoid hard-coding the names of [components] on which the [component under test] depends into our code because static binding severely limits our options regarding how the software is configured as it runs. Instead, we hard-code [the] name of a “component broker” that returns a ready-to-use [component]. The component broker provides some means for the client software … to tell the [component under test] which objects to use for each component request.”

As we shall see, the service locator will be the hard-coded name of the component broker that supplies a ready-to-use component to provide the service. When run in production mode the service locator will be configured to supply dependencies applicable to a production run, but when running the automated unit tests it will be configured to supply those dependencies using test doubles. Accordingly, when the production code of the program is retrofitted to use the service locator to supply its dependencies, it is oblivious to whether it is running in production mode or unit test mode, simply using whatever service is supplied to it by the service locator.

At this point the new class is very brief and capable of handling only a single service, one to facilitate a flights organizer service. In subsequent exercises it will be changed to become capable of handling other services as well.

Notice that the attribute flights_organizer of the service locator is defined as a reference to an interface, not a reference to a class. As a consequence, this attribute may hold a reference to an instance of any class that implements that interface.

This new class will act as the “registry of record” for services required by this program. It already suffers from two weakness:

1. It is defined as a singleton class, and as we’ve noted before, singleton classes come with their own baggage rendering them undesirable entities in some situations.
2. Its sole instance member – attribute flights_organizer – is defined in the public section, meaning that this attribute is available to be changed by external entities.

Let’s register in our issues list that this version introduces new issues #37 and #38.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare

#	Identified	Resolved	Description
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command

#	Identified	Resolved	Description
			(write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A		Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A		Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B		Fourth singleton class is introduced: flights_report_test_double
35	ZAUT302D	ZAUT302N	Unit test method show_flights of class tester explicitly calls test double
36	ZAUT302J	ZAUT302M	Use of singleton classes result in interacting tests
37	**ZAUT401A**		**Fifth singleton class is introduced: service_locator**
38	**ZAUT401A**		**Class service_locator defines externally changeable public attribute flights_organizer**

16.2 Exercise 102

Program: ZAUT401B

Requirements

Reason for change

- Change production code to make use of service locator for flights_organizer.

Changes to be applied

1. Place the following statement at the end of the classic ABAP initialization block:

```
service_locator=>singleton->flights_organizer
                            = flights_organizer=>singleton.
```

2. In the classic ABAP "at selection-screen" block, replace the statement ...

```
call method flights_organizer=>singleton->get_flights_via_carrier
```

… with ...

```
call method service_locator=>singleton->flights_organizer->get_flights_via_carrier
```

… and replace the statement ...

```
if flights_organizer=>singleton->get_flights_count( ) le 00.
```

… with:

```
if service_locator=>singleton->flights_organizer->get_flights_count( ) le 00.
```

3. In subroutine present_report, replace the statement ...

```
flights_stack                = flights_organizer=>singleton->flights_stack.
```

... with ...

```
flights_stack                = service_locator=>singleton->flights_organizer->flights_stack.
```

... and replace the statement ...

```
flights_count                = flights_organizer=>singleton->get_flights_count( ).
```

… with:

```
flights_count                = service_locator=>singleton->flights_organizer->get_flights_count( ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class triggers runtime error for class tester; Status message indicates Processed: 1 program, 1 test classes, **0 test methods**.

Remarks

With this version we have changed the production code to make use of the service locator for gaining access to a flights organizer instead of directly accessing the singleton flights_organizer class, and have included a new statement in the “initialization” classic event block to register such a service when the program begins executing. Whereas running the program in production mode works as expected, the unit test fails because we made no

arrangements for the service locator to register a service for flights_organizer, as we did when we placed a statement in the “initialization” classic event block to facilitate this for the production code.

16.3 Exercise 103

Program: ZAUT401C

Requirements

Reason for change

- Fix the unit test failure encountered in the previous version.

Changes to be applied

1. In method setup of class tester, after line ...

```
flights_organizer=>singleton  = flights_organizer_test_double=>singleton.
```

... add the following lines ...

```
" Register service_locator=>singleton->flights_organizer in service locator:
service_locator=>singleton->flights_organizer
                          = flights_organizer=>singleton.
clear flights_organizer=>singleton. " Prevent direct use of this singleton during test
```

… then perform a global edit changing flights_organizer=>singleton with service_locator=>singleton->flights_organizer on all lines after the lines just added.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have changed the unit test code of class tester to make use of the service locator for gaining access to a flights organizer instead of directly accessing the singleton flights_organizer class. We also have included a new statement in method setup of class tester to register such a service before each unit test begins executing.

Once again, all unit tests pass.

16.4 Exercise 104

Program: ZAUT401D

Requirements

Reason for change

- Introduce service locator capability for flights_report.

Changes to be applied

1. Add the following instance attribute to the public section of class service_locator:

```
data        : o
              o
              o
            , flights_report type ref
                              to flights_reportable
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have expanded the service locator to provide for managing a flights_report service. Again, notice that the new attribute flights_report of the service locator is defined as a reference to an interface, not a reference to a class. As a consequence, this attribute may hold a pointer to an instance of any class that implements that interface. Its new instance member – attribute flights_report – is defined in the public section, meaning that, like existing attribute flights_organizer, this attribute is available to be changed by external entities.

Let’s register in our issues list that this version introduces new issue #39.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue

#	Identified	Resolved	Description
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with

#	Identified	Resolved	Description
			ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A		Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A		Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B		Fourth singleton class is introduced: flights_report_test_double
35	ZAUT302D	ZAUT302N	Unit test method show_flights of class tester explicitly calls test double
36	ZAUT302J	ZAUT302M	Use of singleton classes result in interacting tests
37	ZAUT401A		Fifth singleton class is introduced: service_locator
38	ZAUT401A		Class service_locator defines externally changeable public attribute flights_organizer
39	**ZAUT401D**		**Class service_locator defines externally changeable public attribute flights_report**

16.5 Exercise 105

Program: ZAUT401E

Requirements

Reason for change

- Change production code to make use of service locator for flights_report.

Changes to be applied

1. Place the following statement at the end of the initialization block:

```
service_locator=>singleton->flights_report
```

```
                                = flights_report=>singleton.
```

2. In subroutine present_report, replace the statement ...

```
call method flights_report=>singleton->show_flights
```

... with:

```
call method service_locator=>singleton->flights_report->show_flights
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class triggers runtime error for method present_report; Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have changed the production code to make use of the service locator for gaining access to a flights report instead of directly accessing the singleton flights_report class, and have included a new statement in the “initialization” classic event block to register such a service when the program begins executing. Whereas running the program in production mode works as expected, the unit test fails because we made no arrangements for the service locator to register a service for flights_report, as we did when we placed a statement in the “initialization” classic event block to facilitate this for the production code.

16.6 Exercise 106

Program: ZAUT401F

Requirements

Reason for change

- Fix the unit test failure encountered in the previous version.

Changes to be applied

1. In method setup of class tester, after line …

```
flights_report=>singleton  = flights_report_test_double=>singleton.
```

… add the following lines ...

```
" Register service_locator=>singleton->flights_report in service locator:
service_locator=>singleton->flights_report
                           = flights_report=>singleton.
clear flights_report=>singleton. " Prevent direct use of this singleton during test
```

... then perform a global edit changing flights_report=>singleton with service_locator=>singleton->flights_report on all lines after the lines just added.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have changed the unit test code of class tester to make use of the service locator for gaining access to a flights report instead of directly accessing the singleton flights_report class. We also have included a new statement in method setup of class tester to register such a service before each unit test begins executing.

Once again, all unit tests pass.

16.7 Exercise 107

Program: ZAUT401G

Requirements

Reason for change

- Remove capability for external entities to directly change public attributes of class service_locator.

Changes to be applied

1. Define new interface service_locatable after interface flights_reportable:

```
interface service_locatable.
    methods      : register_flights_organizer
                     importing
                       flights_organizer
                         type ref
                           to flights_organizable
                 , register_flights_report
                     importing
                       flights_report
                         type ref
                           to flights_reportable
                 .
endinterface.
```

2. Include the following statements after the public section statement in class service_locator:

```
    interfaces   : service_locatable
                 .
    aliases      : register_flights_organizer
                     for service_locatable~register_flights_organizer
                 , register_flights_report
```

```
                for service_locatable~register_flights_report
            .
```

3. In class service_locator, apply the "read-only" qualifier to attributes flights_organizer and flights_report.
4. Add the following method implementations at the end of class service_locator:

```
method register_flights_organizer.
  me->flights_organizer         = flights_organizer.
endmethod.
method register_flights_report.
  me->flights_report            = flights_report.
endmethod.
```

5. In the initialization block replace these statements ...

```
  service_locator=>singleton->flights_organizer
                                = flights_organizer=>singleton.
  service_locator=>singleton->flights_report
                                = flights_report=>singleton.
```

… with these statements:

```
  service_locator=>singleton->register_flights_organizer(
    exporting
      flights_organizer         = flights_organizer=>singleton
    ).
  service_locator=>singleton->register_flights_report(
    exporting
      flights_report            = flights_report=>singleton
    ).
```

6. In method setup of class tester, replace each of the statements ...

```
  service_locator=>singleton->flights_organizer
                                = flights_organizer=>singleton.

  service_locator=>singleton->flights_report
                                = flights_report=>singleton.
```

… with their counterpart statements:

```
  service_locator=>singleton->register_flights_organizer(
    exporting
      flights_organizer         = flights_organizer=>singleton
    ).

  service_locator=>singleton->register_flights_report(
    exporting
      flights_report            = flights_report=>singleton
    ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have prevented external entities from having the ability to directly change the public attributes of class service_locator holding the service references. Locations in the code formerly making direct changes to the public attributes of class service_locator have been replaced with calls to methods of the service locator to have the service registered.

This conforms with best practices of object-oriented design which discourages direct change access to the values of public attributes by external entities. Instead, the class provides its own public methods to enable external entities to request a change to the public attributes. It enables the class to retain full control over its state (the values of its attributes) because the class itself is now in control over any changes made to its public attributes.

Let's register in our issues list that this version resolves issues #38 and #39.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields

#	Identified	Resolved	Description
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A		Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A		Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B		Fourth singleton class is introduced: flights_report_test_double
35	ZAUT302D	ZAUT302N	Unit test method show_flights of class tester explicitly calls test double

#	Identified	Resolved	Description
36	ZAUT302J	ZAUT302M	Use of singleton classes result in interacting tests
37	ZAUT401A		Fifth singleton class is introduced: service_locator
38	**ZAUT401A**	**ZAUT401G**	**Class service_locator defines externally changeable public attribute flights_organizer**
39	**ZAUT401D**	**ZAUT401G**	**Class service_locator defines externally changeable public attribute flights_report**

17 ABAP Unit Testing 402 – Introducing a Service Factory

This section describes the requirements for the exercise programs associated with the Chapter 9 section titled Using a Service Factory in the book Automated Unit Testing with ABAP.

17.1 Exercise 108

Program: ZAUT402A

Requirements

Reason for change

- Begin the process of providing a factory to create and register services.

Changes to be applied

1. Define new interface service_creatable after interface flights_reportable:

```
interface service_creatable.
    methods      : create_all_services
                 , create_flights_organizer
                 , create_flights_report
                 .
endinterface.
```

2. Define new class service_factory after class flights_report_test_double:

```
class service_factory                 definition
                                      final
                                      create private
                                      .
  public section.
    interfaces   : service_creatable
                 .
    aliases      : create_all_services
                     for service_creatable~create_all_services
                 , create_flights_organizer
                     for service_creatable~create_flights_organizer
                 , create_flights_report
                     for service_creatable~create_flights_report
                 .
    class-data   : singleton      type ref
                                    to service_factory
                                         read-only
                 .
    class-methods: class_constructor
                 .
endclass.
class service_factory                 implementation.
  method class_constructor.
    create object service_factory=>singleton.
  endmethod.
  method create_all_services.
    me->create_flights_organizer( ).
    me->create_flights_report( ).
  endmethod.
  method create_flights_organizer.
  endmethod.
  method create_flights_report.
  endmethod.
endclass.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have introduced a service locator factory, yet another singleton class that eventually will become capable of creating service entities and then calling the service locator to register these services. For now it provides 3 capabilities:

1. create a flights_organizer service
2. create a flights_report service
3. create all services

Its capability to create all services is simply a single public method that calls all of the other public methods for creating specific services. None of the methods for creating a specific service have any implementation at this point.

Let’s register in our issues list that this version introduces new issue #40.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count

#	Identified	Resolved	Description
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A		First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A		Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with

#	Identified	Resolved	Description
			ZAUT202E; see issue #29)
32	ZAUT301A		Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B		Fourth singleton class is introduced: flights_report_test_double
35	ZAUT302D	ZAUT302N	Unit test method show_flights of class tester explicitly calls test double
36	ZAUT302J	ZAUT302M	Use of singleton classes result in interacting tests
37	ZAUT401A		Fifth singleton class is introduced: service_locator
38	ZAUT401A	ZAUT401G	Class service_locator defines externally changeable public attribute flights_organizer
39	ZAUT401D	ZAUT401G	Class service_locator defines externally changeable public attribute flights_report
40	**ZAUT402A**		**Sixth singleton class is introduced: service_factory**

17.2 Exercise 109

Program: ZAUT402B

Requirements

Reason for change

- Complete the process of providing a factory to create and register services.

Changes to be applied

1. Remove the "create private" statement from class flights_organizer.
2. Remove the "create private" statement from class flights_report.
3. Include the following statements as the implementation for method create_flights_organizer of class service_factory:

```
data        : flights_organizer
                          type ref
                            to flights_organizable
            .
create object flights_organizer
         type flights_organizer.
service_locator=>singleton->register_flights_organizer(
  exporting
    flights_organizer         = flights_organizer
    ).
```

4. Include the following statements as the implementation for method create_flights_report of class service_factory:

```
data         : flights_report
                              type ref
                                to flights_reportable
               .
create object flights_report
         type flights_report.
service_locator=>singleton->register_flights_report(
  exporting
    flights_report            = flights_report
    ).
```

5. In the initialization event block, replace the statements ...

```
service_locator=>singleton->register_flights_organizer(
  exporting
    flights_organizer         = flights_organizer=>singleton
  ).
service_locator=>singleton->register_flights_report(
  exporting
    flights_report            = flights_report=>singleton
  ).
```

... with the single statement:

```
service_factory=>singleton->create_all_services( ).
```

6. Replace the implementation of method setup of class tester with the following statements:

```
service_factory=>singleton->create_all_services( ).
" To prevent the possibility of interacting tests, refresh all singleton objects:
create object flights_organizer_test_double=>singleton
         type flights_organizer_test_double.
create object flights_report_test_double=>singleton
         type flights_report_test_double.
" Register flights_organizer_test_double=>singleton as flights_organizer service:
service_locator=>singleton->register_flights_organizer(
  exporting
    flights_organizer         = flights_organizer_test_double=>singleton
  ).
clear flights_organizer=>singleton. " Prevent direct use of this singleton during test
" Register flights_report_test_double=>singleton as flights_report service:
service_locator=>singleton->register_flights_report(
  exporting
    flights_report            = flights_report_test_double=>singleton
  ).
clear flights_report=>singleton. " Prevent direct use of this singleton during test
carrier                       = american_airlines.
call method service_locator=>singleton->flights_organizer->get_flights_via_carrier
  exporting
    carrier                   = carrier
    .
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have removed the “create private” from the class definitions for classes flights_organizer and flights_report, meaning that, by default, instances of these classes can be created by any external entity. Indeed, this is coupled with implementations in methods create_flights_organizer and create_flights_report of class service_factory to (a) create a new instance of the respective class and (b) call the service locator to register the new instance as the corresponding service.

Effectively, class service_factory becomes the external entity creating instances of classes flights_organizer and flights_report. Until we removed the “create private” clauses from their respective class statements, classes flights_organizer and flights_report could not have been instantiated by any entity other than themselves (see note below).

We have removed from the production code all direct calls to methods of class service_locator – both of which appeared in the “initialization” classic event block – and replaced them with a single call to method create_all_services of class service_factory. We’ve also modified method setup of class tester also to call to method create_all_services of class service_factory.

Note: Prior to removing their “create private” clauses, classes flights_organizer and flights_report technically could have been instantiated by any class implementing the interfaces associated with their respective “friends” clauses. At this point only unit test class tester implements those respective interfaces.

17.3 Exercise 110

Program: ZAUT402C

Requirements

Reason for change

- Reduce number of singleton classes.

Changes to be applied

1. Remove the singleton attributes and class_constructor methods from these classes:
   - flights_organizer
   - flights_organizer_test_double
   - flights_report
   - flights_report_test_double
2. Replace the implementation of method setup of class tester with the following statements:

```
data        : flights_organizer_test_double
                        type ref
                          to flights_organizable
            , flights_report_test_double
                        type ref
                          to flights_reportable
            .
service_factory=>singleton->create_all_services( ).
" To prevent the possibility of interacting tests, refresh all singleton objects:
create object flights_organizer_test_double
         type flights_organizer_test_double.
create object flights_report_test_double
         type flights_report_test_double.
" Register flights_organizer_test_double=>singleton as flights_organizer service:
service_locator=>singleton->register_flights_organizer(
  exporting
    flights_organizer         = flights_organizer_test_double
  ).
```

```
" Register flights_report_test_double=>singleton as flights_report service:
service_locator=>singleton->register_flights_report(
  exporting
    flights_report            = flights_report_test_double
  ).
carrier                       = american_airlines.
call method service_locator=>singleton->flights_organizer->get_flights_via_carrier
  exporting
    carrier                   = carrier
    .
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have removed the singleton attributes and the corresponding class_constructor methods from these classes:

- flights_organizer
- flights_organizer_test_double
- flights_report
- flights_report_test_double

They no longer represent singleton classes. At this point the only singleton classes remaining are service_locator and service_factory.

Notice that we’ve changed method setup of class tester to reflect the removal of the singleton attributes from the test double classes.

Note: Classes flights_organizer_test_double and flights_report_test_double still have the “create private” clause on their class definition statements, but instances of these classes already have been created by class tester by virtue of it implementing the interface named on the “friends” clause of these classes.

Note: Now that these classes no longer represent singleton classes it is possible to create multiple instances of them. Despite this capability, only a single instance of each is created by the program because only one instance of each class can be registered by the service locator.

Let’s register in our issues list that this version resolves issues #27, #30, #32 and #34.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount

#	Identified	Resolved	Description
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)

#	Identified	Resolved	Description
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	**ZAUT201A**	**ZAUT402C**	**First singleton class is introduced: flights_organizer**
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	**ZAUT202A**	**ZAUT402C**	**Second singleton class is introduced: flights_report**
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	**ZAUT301A**	**ZAUT402C**	**Third singleton class is introduced: flights_organizer_test_double**
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	**ZAUT302B**	**ZAUT402C**	**Fourth singleton class is introduced: flights_report_test_double**
35	ZAUT302D	ZAUT302N	Unit test method show_flights of class tester explicitly calls test double
36	ZAUT302J	ZAUT302M	Use of singleton classes result in interacting tests
37	ZAUT401A		Fifth singleton class is introduced: service_locator
38	ZAUT401A	ZAUT401G	Class service_locator defines externally changeable public attribute flights_organizer
39	ZAUT401D	ZAUT401G	Class service_locator defines externally changeable public attribute flights_report
40	ZAUT402A		Sixth singleton class is introduced: service_factory

17.4 Exercise 111

Program: ZAUT402D

Requirements

Reason for change

- Change production code to use test data.

Changes to be applied

1. Include the following interface on the interfaces statement in class service_factory:

```
interfaces   : o
               o
               o
             , flights_organizer_testable
```

2. In method create_flights_organizer of class service_factory, replace the statement ...

```
create object flights_organizer
         type flights_organizer.
```

… with the statement:

```
create object flights_organizer
         type flights_organizer_test_double.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria, but notice that the list of flights is the test data list produced by class flights_organizer_test_double.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have changed the service factory to create an instance of flights_organizable_test_double to be registered as the flights_organizer service. Running the program produces the ALV report but it contains test data records generated by class flights_organizable_test_double.

We might consider this to be a significant exposure to writing production code since here we see that a class intended solely to act as a unit test double can be used by the production code. The next exercise shows how we can eliminate this exposure.

17.5 Exercise 112

Program: ZAUT402E

Requirements

Reason for change

- Insure production code cannot use test doubles.

Changes to be applied

1. Apply the "for testing" clause after the create private clauses in the definition statements of classes flights_organizer_test_double and flights_report_test_double:

```
                                for testing
```

2. Check syntax: method create_flights_organizer of class service_factory diagnosed with the following error message:

   "The reference to a test class (identification with FOR TESTING) is only possible in test classes."

   The compiler will not allow the execution path of the program to refer to a class marked as a test class because test classes will not be made available in production.

3. Remove from class service_factory the following statement:

```
 interfaces flights_organizer_testable.
```

4. Reset method create_flights_organizer of class service_factory to create an object of type flights_organizer:

```
create object flights_organizer
         type flights_organizer.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria, and notice that the list of flights is from the SFLIGHT table.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 1 test classes, 9 test methods.

Remarks

With this version we have added the “for testing” clause to the class definition statements of classes flights_organizer_test_double and flights_report_test_double and removed the changes implemented in the previous version.

As we can see, once we applied the “for testing” clause to the class definition statements of classes flights_organizer_test_double and flights_report_test_double, the syntax was rendered invalid due to method create_flights_organizer of class service_factory containing a statement to create an instance of class flights_organizer_test_double. The syntax checker detects that there are references in the production code (service_factory) to components defined for use only during the automated unit tests (flights_organizer_test_double). Indeed, the compiler needs to have this capability because components marked “for testing” do not get compiled into production, so if production code could reference components marked “for testing”, the end result would be a failed compile in production.

17.6 Exercise 113

Program: ZAUT402F

Requirements

Reason for change

- Begin the process of providing unit tests for class service_factory.

Changes to be applied

1. Create new test class service_factory_autester for testing the service factory class. Place it at the end of the program using the following code:

```
class service_factory_autester         definition
                                       final
                                       for testing
                                       risk level harmless
                                       duration short
                                       .
  private section.
    methods      : setup
                 , create_all_services
                     for testing
                 , create_flights_organizer
                     for testing
                 , create_flights_report
                     for testing
                 .
endclass.
class service_factory_autester         implementation.
  method setup.
  endmethod.
  method create_all_services.
    cl_abap_unit_assert=>fail(
      msg                        = 'Test method not implemented'
      ).
  endmethod.
  method create_flights_organizer.
    cl_abap_unit_assert=>fail(
      msg                        = 'Test method not implemented'
      ).
  endmethod.
  method create_flights_report.
    cl_abap_unit_assert=>fail(
      msg                        = 'Test method not implemented'
      ).
  endmethod.
endclass.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class triggers errors for all methods of class service_factory_autester; Status message indicates Processed: 1 program, 2 test classes, 12 test methods.

Remarks

With this version we have introduced a new unit test class for testing class service_factory. Currently this unit test is defined with unit test methods for create_flights_organizer, create_flights_report and create_all_services, the same names as the corresponding public methods of class service_factory. In each case the unit test method implementation simply issues an error message that there is no implementation provided for the test method.

Notice that now there are two classes defined for unit testing, a fact reflected in the status message produced by the test runner at the completion of running the unit tests.

Note: The name of the new unit test class is the same name as the class under test (CUT) it is intended to test, with the added suffix "_autester", for **A**BAP **U**nit **Tester**. Over the years I have adopted a unit test class naming convention that would end with the suffix "_autester" and be preceded by as many characters of the corresponding class under test as would fit within the 30-character name limit, sacrificing trailing characters of the CUT name as would be required. This naming convention is a reflection of the suffix "_testable" I adopted for names of interfaces I had defined for applying to the "friends" clause of a class definition statement to enable a corresponding unit test class to access its private members via implementing that interface. This is not a suggestion to use this naming convention with your own automated unit tests but simply a clarification on how these suffixes play a part in documenting the names of components used by these exercise programs.

17.7 Exercise 114

Program: ZAUT402G

Requirements

Reason for change

- Complete the process of providing unit tests for class service_factory.

Changes to be applied

1. In method setup of class service_factory_autester, replace the assert fail statement with the following statements:

```
cl_abap_unit_assert=>assert_not_bound(
  act                       = service_locator=>singleton->flights_organizer
  ).
cl_abap_unit_assert=>assert_not_bound(
  act                       = service_locator=>singleton->flights_report
  ).
```

2. In method create_all_services of class service_factory_autester, replace the assert fail statement with the following statements:

```
service_factory=>singleton->create_all_services( ).
cl_abap_unit_assert=>assert_bound(
  act                       = service_locator=>singleton->flights_organizer
  ).
cl_abap_unit_assert=>assert_bound(
  act                       = service_locator=>singleton->flights_report
  ).
```

3. In method create_flights_organizer of class service_factory_autester, replace the assert fail statement with the following statements:

```
service_factory=>singleton->create_flights_organizer( ).
cl_abap_unit_assert=>assert_bound(
  act                        = service_locator=>singleton->flights_organizer
  ).
cl_abap_unit_assert=>assert_not_bound(
  act                        = service_locator=>singleton->flights_report
  ).
```

4. In method create_flights_report of class service_factory_autester, replace the assert fail statement with the following statements:

```
service_factory=>singleton->create_flights_report( ).
cl_abap_unit_assert=>assert_not_bound(
  act                        = service_locator=>singleton->flights_organizer
  ).
cl_abap_unit_assert=>assert_bound(
  act                        = service_locator=>singleton->flights_report
  ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class passes for one test but triggers two errors during method setup; Status message indicates Processed: 1 program, 2 test classes, 10 test methods.

Remarks

With this version we have provided implementations for each of the methods of unit test class service_factory_autester. Notice that method setup now asserts that none of the services offered by class service_locator are bound. Notice also that method create_all_services of class service_factory_autester calls method create_all_services of class service_factory. In each case these unit test methods assert that the corresponding service is bound or not bound as applicable.

Note: The approach taken so far with the unit tests in this program is that the production code existed first and then a unit test class and its methods was provided afterward. This was the case for the unit tests defined to test each of the existing subroutines in the program as well as with unit test class service_factory_autester, to test class service_factory, written after class service_factory already was implemented. We have been taking this approach to become familiar with the process of retrofitting an existing program with new unit tests, a task certain to be applicable to everyone performing this set of exercises.

There is, however, a different approach that should be considered when writing a new ABAP component and its corresponding unit tests. It is known as Test Driven Development, usually abbreviated simply as TDD, where a failing unit test is written first followed by the minimal production code to make the failing unit test pass. We will explore this approach with subsequent exercises.

17.8 Exercise 115

Program: ZAUT402H

Requirements

Reason for change

- Fix the unit test failures encountered in the previous version.

Changes to be applied

1. Define new empty interface service_locator_testable following interface service_creatable using the following statements:

```
interface service_locator_testable.
endinterface.
```

2. Apply the "friends service_locator_testable" statement to class service_locator after the "create private" statement:

```
                                    friends service_locator_testable
```

3. Create new class service_locator_test_helper ahead of class tester using the following statements:

```
class service_locator_test_helper      definition
                                       final
                                       for testing
                                       .
  public section.
    interfaces   : service_locator_testable
                 .
    methods      : clear_all_service_locators
                 .
endclass.
class service_locator_test_helper      implementation.
  method clear_all_service_locators.
    clear: service_locator=>singleton->flights_organizer
         , service_locator=>singleton->flights_report
         .
    cl_abap_unit_assert=>assert_not_bound(
      act                          = service_locator=>singleton->flights_organizer
      ).
    cl_abap_unit_assert=>assert_not_bound(
      act                          = service_locator=>singleton->flights_report
      ).
  endmethod.
endclass.
```

4. Replace the statements in method setup of class service_factory_autester with the following statements:

```
    data         : service_locator_test_helper
                                    type ref
                                      to service_locator_test_helper
                 .
    create object service_locator_test_helper.
    service_locator_test_helper->clear_all_service_locators( ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 12 test methods.

Remarks

With this version we've introduced a new class to help with testing the service locator. Notice that this new class service_locator_test_helper implements empty interface service_locator_testable, which class service_locator now includes on a "friends" clause of its class definition statement. This is how class service_locator_test_helper gains access to the private attributes of class service_locator.

Notice also that new class service_locator_test_helper is marked "for testing", and that its sole method clear_all_service_locators contains calls to assertion methods of class cl_abap_unit_assert, but that none of its own methods are marked "for testing".

Notice that its public method clear_all_service_locators is called by method setup of class service_factory_autester, but despite being marked "for testing" this class does not contribute to the number of test classes appearing in the status message after the automated unit test run has completed.

In addition, notice that method setup of class service_factory_autester formerly asserted that none of the service_locator attributes are bound, a task that now has been delegated to this new class. It has the beneficial effect of keeping method setup of class service_factory_autester free from requiring changes to perform any additional assertions as new service locators are added to the service locator class.

Note: At this point it can be argued that relieving method setup of class service_factory_autester of the requirement to perform any additional assertions as new service locators are added to the service locator class merely shifts this requirement to this new test helper class. Whereas that is true, we shall see in subsequent exercises that this helper class also will provide this service for other unit test classes, not just for service_factory_autester. Accordingly, had it not been moved to a helper class, then the assertions it performs would had to have been duplicated in the setup methods for all classes calling this new helper class, requiring each of them to be changed when a new service is added to the service locator.

A class defined for the purpose of providing services to other unit test classes is regarded as a "Test Helper", a unit testing pattern cataloged by Gerard Meszaros (xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 643).

18 ABAP Unit Testing 501 – Gaining Control Over Global Class Dependencies

This section describes the requirements for the exercise programs associated with the Chapter 10 section titled Using the Service Locator to Manage Global Classes in the book Automated Unit Testing with ABAP.

18.1 Exercise 116

Program: ZAUT501A

Requirements

Reason for change

- Create unit test for testing presence of service to calculate flight revenue.

Changes to be applied

1. Add new test method create_revenue_calculator to class service_factory_autester:
   - Add method definition for create_revenue_calculator to the private section of class service_factory_autester after the definition for method create_flights_report:

```
methods      : o
               o
               o
             , create_revenue_calculator
                 for testing
```

   - Include the following method implementation after the implementation for method create_flights_report:

```
method create_revenue_calculator.
  cl_abap_unit_assert=>fail(
    msg                          = 'Test method not implemented'
    ).
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class service_factory_autester triggers error during method create_revenue_calculator; Status message indicates Processed: 1 program, 2 test classes, 13 test methods.

Remarks

With this version we have defined new unit test method create_revenue_calculator to class service_factory_autester. This is being done in preparation for having the service locator also to manage the service for calculating flight revenue. This service currently is provided to our program by global class zcl_flight_revenue_calculator when subroutine adjust_flight_revenue directly calls its method get_flight_revenue.

Global class zcl_flight_revenue_calculator represents an example of a component on which subroutine adjust_flight_revenue is dependent. Such a component is known as a *depended-on component,* also known by the acronym *DOC*. It is only by gaining control over depended-on components that we can rely on the results of our unit tests.

The unit test fails because new unit test create_revenue_calculator of class service_factory_autester simply has a call to method fail of class cl_abap_unit_assert with a message that no test implementation has been provided.

18.2 Exercise 117

Program: ZAUT501B

Requirements

Reason for change

- Fix the unit test failure encountered in the previous version.

Changes to be applied

1. Add to class service_locator new public attribute revenue_calculator, after the attribute flights_report:

```
data         : o
               o
               o
             , revenue_calculator
                             type ref
                               to zcl_flight_revenue_calculator
                                    read-only
```

2. In method clear_all_service_locators of class service_locator_test_helper
   - Add the following line to the clear statement:

```
clear: o
       o
       o
     , service_locator=>singleton->revenue_calculator
```

   - At end of this method add an assertion to test that this attribute is not bound:

```
cl_abap_unit_assert=>assert_not_bound(
  act                        = service_locator=>singleton->revenue_calculator
  ).
```

3. Add to end of method create_all_services of class service_factory_autester an assertion to test that attribute service_locator=>singleton->revenue_calculator is bound:

```
cl_abap_unit_assert=>assert_bound(
  act                        = service_locator=>singleton->revenue_calculator
  ).
```

4. Add to end of methods create_flights_organizer and create_flights_report of class service_factory_autester an assertion to test that attribute service_locator=>singleton->revenue_calculator is not bound:

```
cl_abap_unit_assert=>assert_not_bound(
  act                       = service_locator=>singleton->revenue_calculator
  ).
```

5. Replace failure assertion in method create_revenue_calculator of class service_factory_autester with the following code:

```
cl_abap_unit_assert=>assert_not_bound(
  act                       = service_locator=>singleton->flights_organizer
  ).
cl_abap_unit_assert=>assert_not_bound(
  act                       = service_locator=>singleton->flights_report
  ).
cl_abap_unit_assert=>assert_bound(
  act                       = service_locator=>singleton->revenue_calculator
  ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class service_factory_autester triggers error during methods create_all_services and create_revenue_calculator; Status message indicates Processed: 1 program, 2 test classes, 13 test methods.

Remarks

With this version we’ve added a new public attribute to class service_locator, one for retaining a service for a flight revenue calculator. With this new attribute it is now capable of managing 3 services.

We’ve also changed class service_locator_test_helper to clear and assert so on this new attribute of class service_locator. We’ve also adjusted all the unit test methods of class service_factory_autester to include an assertion of bound or not bound, as applicable, for this new service, including its new unit test method create_revenue_calculator introduced with the previous version.

The unit test still fails because even though unit test method create_revenue_calculator of class service_factory_autester now applies expected bound/not bound assertions on the attributes of class service_locator, there is no code executed during the unit test to register a service for a flights revenue calculator.

18.3 Exercise 118

Program: ZAUT501C

Requirements

Reason for change

- Fix the unit test failures encountered in the previous version.

Changes to be applied

1. Via SE24, create new global interface zif_flight_revenue_calculable:
   - Properties:
     - Description Fights revenue calculable
     - Package $TMP
     - Unicode checks active checked
   - Methods:
     - GET_FLIGHT_REVENUE - Instance method

Parameter	Type	Pass Value	Optional	Typing Method	Associated Type	Default Value
FARE_PRICE	Importing	unchecked	unchecked	Type	S_PRICE	(blank)
NUMBER_OF_PASSENGERS	Importing	unchecked	unchecked	Type	S_SEATSOCC	(blank)
FLIGHT_REVENUE	Exporting	unchecked	unchecked	Type	S_SUM	(blank)

   Activate all global interface components.

2. Change global class zcl_flight_revenue_calculator to implement global interface zif_flight_revenue_calculable:
   - Using the source-code based editor, replace the public section with the following code:

```
interfaces     : zif_flight_revenue_calculable
                 .
aliases        : get_flight_revenue
                   for zif_flight_revenue_calculable~get_flight_revenue
                 .

class-data singleton type ref to zif_flight_revenue_calculable.

class-methods class_constructor.
```

   - Using the source-code based editor, change the class_constructor implementation from ...

```
  create object singleton.
```

     … to:

```
  create object singleton type zcl_flight_revenue_calculator.
```

   Activate all global class components.

3. Add the following method definition to interface service_locatable, to follow the definition for register_flights_report:

```
  methods      : o
                 o
                 o
               , register_revenue_calculator
                   importing
                     revenue_calculator
                       type ref
                         to zif_flight_revenue_calculable
```

4. Add the following method definition to interface service_creatable, to follow the definition for create_flights_report:

```
  methods      : o
                 o
                 o
               , create_revenue_calculator
```

5. Add the following to the aliases statement of class service_locator after the one for register_flights_report:

```
  aliases      : o
```

```
                o
                o
              , register_revenue_calculator
                  for service_locatable~register_revenue_calculator
```

6. In class service_locator, change attribute revenue_calculator from type zcl_flight_revenue_calculator to type zif_flight_revenue_calculable.
7. Add the following method implementation to class service_locator after the one for register_flights_report:

```
method register_revenue_calculator.
  me->revenue_calculator        = revenue_calculator.
endmethod.
```

8. Add the following to the aliases statement of class service_factory after the one for create_flights_report:

```
  aliases      : o
                 o
                 o
               , create_revenue_calculator
                   for service_creatable~create_revenue_calculator
```

9. Add the following statement to the end of method create_all_services of class service_factory:

```
  me->create_revenue_calculator( ).
```

10. Add the following method implementation to class service_factory after the one for create_flights_report:

```
method create_revenue_calculator.
  data         : revenue_calculator
                                type ref
                                  to zif_flight_revenue_calculable
               .
  try.
    revenue_calculator          ?= zcl_flight_revenue_calculator=>singleton.
  catch cx_sy_move_cast_error.
  endtry.
  service_locator=>singleton->register_revenue_calculator(
    exporting
      revenue_calculator        = revenue_calculator
      ).
endmethod.
```

11. Add the following statement to the start of method create_revenue_calculator of class service_factory_autester:

```
  service_factory=>singleton->create_revenue_calculator( ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 13 test methods.

Remarks

Here we have defined a new global interface zif_flight_revenue_calculable and changed global class zcl_flight_revenue_calculator to implement that new global interface. This is a necessary step in the quest to enable this global class to be accessed through a variable defined as a reference to an interface. It will make it possible for us to substitute this global class with a different class that implements the same global interface.

With this version of the exercise program we have changed the type of the revenue calculator attribute of class service_locator to refer to an interface instead of a specific class and provided the class with a public method for registering a flight revenue calculator, provided class service_factory with a method to create and register a flight revenue calculator and included a call to this new method from within method create_all_services, and finally, to enable the failing unit test to pass, have included in unit test create_revenue_calculator of class service_factory_autester a call to the new method of the service factory to create and register a flight revenue calculator.

18.4 Exercise 119

Program: ZAUT501D

Requirements

Reason for change

- Use service locator to locate flight revenue calculation service.

Changes to be applied

1. In subroutine adjust_flight_revenue, replace statement ...

```
call method zcl_flight_revenue_calculator=>singleton->get_flight_revenue
```

... with:

```
call method service_locator=>singleton->revenue_calculator->get_flight_revenue
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 13 test methods.

Remarks

With this version we have changed subroutine adjust_flight_revenue so that it uses the service locator to access the service provider for calculating flight revenue instead of directly accessing the singleton zcl_flight_revenue_calculator class. With this change the service locator is managing 3 services used by both the production code and the unit test code.

18.5 Exercise 120

Program: ZAUT501E

Requirements

Reason for change

- Method adjust_flight_revenue should not know how method get_flight_revenue of class zcl_flight_revenue_calculator was implemented, but it effectively duplicates the implementation. Use a known expected value to test subroutine adjust_flight_revenue.

Changes to be applied

1. In method adjust_flight_revenue of class tester, replace statement ...

```
flight_revenue             = flights_entry-price * flights_entry-seatsocc.
```

... with:

```
flight_revenue             = 12345.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers error during methods adjust_flight_revenue; Status message indicates Processed: 1 program, 2 test classes, 13 test methods.

Remarks

With this version we have changed unit test adjust_flight_revenue of class tester to use an expected flight revenue value of some value we have randomly chosen within the unit test itself.

This unit test fails because there is virtually no chance that any of the flight records altered after the call to subroutine adjust_flight_revenue would have the same value randomly chosen to be used in the assertion.

This illustrates a problem often encountered in writing automated unit tests. There should be no reason the unit test needs to know anything about the processing logic used by a called routine in creating the values subsequently used in the unit test assertions. Notice that before this change the unit test used the same logic to determine the flight revenue used by method get_flight_revenue of class zcl_flight_revenue_calculator. It established an uncontrollable dependency on the unit test to know the internal processing logic used to create the values it was provided for making assertions. With such a dependency, any changes to the processing logic in the called procedure could cause the unit test to fail, even when the processing logic change is later determined to be correct. Accordingly, the unit test should be able to determine its own value that should be returned by a called routine, and the preparation for running the test should include gaining control over the depended-on component that provides that value.

18.6 Exercise 121

Program: ZAUT501F

Requirements

Reason for change

- Fix the unit test failure encountered in the previous version.

Changes to be applied

1. Create new global class zcl_revenue_calc_test_double:
   - Via SE24, copy zcl_flight_revenue_calculator to zcl_revenue_calc_test_double:
     - Package: $TMP
   - Using the source-code based editor, define the following new constant in the public section:

```
constants flight_revenue_for_test type sflight-paymentsum value 12345.
```

   - In method class_constructor of new class zcl_revenue_calc_test_double, change create object statement to create object of type zcl_revenue_calc_test_double.
   - Replace implementation of method get_flight_revenue with the following statement:

```
flight_revenue              = flight_revenue_for_test.
```

   - If necessary, in the Class-Relevant Local Definitions, change the reference from zcl_flight_revenue_calculator to zcl_revenue_calc_test_double.

   Activate all global class components.

2. In method adjust_flight_revenue of class tester, do the following:
   - Include the following statement ahead of the perform statement to subroutine adjust_flight_revenue:

```
service_locator=>singleton->register_revenue_calculator(
  exporting
    revenue_calculator        = zcl_revenue_calc_test_double=>singleton
    ).
```

   - Replace statement ...

```
  flight_revenue              = 12345.
```

   ... with

```
  flight_revenue              = zcl_revenue_calc_test_double=>flight_revenue_for_test.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 13 test methods.

Remarks

Here we have defined a new global class zcl_revenue_calc_test_double by copying class zcl_flight_revenue_calculator and changing it to always provide the same value for flight revenue returned by method get_flight_revenue. The flight revenue value returned to the caller is defined as a public constant in this copied class. Accordingly, any unit test directly or indirectly calling method get_flight_revenue of this class simply can reference this public constant in any assertions it would apply on values being returned from it.

Notice that now the unit test no longer decides the value it should use in the assertion but delegates this to the test double to provide. The unit test now passes even though it has no clue how returning values are determined and, more importantly, is no longer aware of the internal processing implemented in any specific class.

With this version we have changed unit test adjust_flight_revenue of class tester to (a) register with the service locator an instance of class zcl_revenue_calc_test_double to provide the flight revenue calculation service and (b) reference the public constant of this class to provide the value that should cause the assertions to pass.

19 ABAP Unit Testing 502 – Gaining Control Over Function Module Dependencies

This section describes the requirements for the exercise programs associated with the Chapter 10 section titled Using the Service Locator to Manage Function Modules in the book Automated Unit Testing with ABAP.

19.1 Exercise 122

Program: ZAUT502A

Requirements

Reason for change

- Change service locator to handle service for calculating airfare discount.

Changes to be applied

1. Include the following instance attribute in the public section of class service_locator after the one for revenue_calculator:

```
data        : o
              o
              o
            , discount_calculator
                           type funcname
```

2. In subroutine apply_flight_discount, change the call function statement from ...

```
call function 'ZCALCULATE_DISCOUNTED_AIRFARE'
```

… to:

```
call function service_locator=>singleton->discount_calculator
```

Run

Action: Specify Airline ‘AA’, **discount 10**, ALV classic list and press Execute.

Result: Program fails with exception CX_SY_DYN_CALL_ILLEGAL_FUNC.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 13 test methods.

Remarks

With this version we’ve added a new public attribute to class service_locator, one for retaining a service for a flight discount calculator. With this new attribute it is now capable of managing 4 services.

At this point this service is being provided by function module CALCULATE_DISCOUNTED_AIRFARE. Accordingly, we are enabling the service locator to retain a service provided by a function module rather than an

instance of a class. This will make it possible for us to substitute this function module with a different one to be registered to the service locator.

This function module represents another example of a depended-on component, in this case one on which subroutine apply_flight_discount is dependent. Subroutine apply_flight_discount was changed to use the function module name supplied by the service locator attribute for discount calculator instead of calling an explicit function module as it had with the previous version.

The program fails when run in production mode due to a missing function module name occupying the new service locator attribute discount_calculator. Worse, the unit test *does not* fail because method apply_flight_discount of class tester does not call subroutine apply_flight_discount with calling parameters that would cause the missing service locator function module to be invoked.

Let's register in our issues list that this version introduces new issue #41.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields

#	Identified	Resolved	Description
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A	ZAUT402C	First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A	ZAUT402C	Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A	ZAUT402C	Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B	ZAUT402C	Fourth singleton class is introduced: flights_report_test_double
35	ZAUT302D	ZAUT302N	Unit test method show_flights of class tester explicitly calls test double

#	Identified	Resolved	Description
36	ZAUT302J	ZAUT302M	Use of singleton classes result in interacting tests
37	ZAUT401A		Fifth singleton class is introduced: service_locator
38	ZAUT401A	ZAUT401G	Class service_locator defines externally changeable public attribute flights_organizer
39	ZAUT401D	ZAUT401G	Class service_locator defines externally changeable public attribute flights_report
40	ZAUT402A		Sixth singleton class is introduced: service_factory
41	**ZAUT502A**		**Method apply_flight_discount of class tester fails to cause subroutine apply_flight_discount to call the function module**

19.2 Exercise 123

Program: ZAUT502B

Requirements

Reason for change

- Change service locator and service factory to accommodate creating and registering a flight discount calculation service.

Changes to be applied

1. Include the following method definition with interface service_locatable after the one for register_revenue_calculator:

```
methods      : o
               o
               o
             , register_discount_calculator
                 importing
                   discount_calculator
                     type funcname
```

2. Include the following method definition with interface service_creatable after the one for create_revenue_calculator:

```
methods      : o
               o
               o
             , create_discount_calculator
```

3. Include the following alias in the public section of class service_locator after the one for register_revenue_calculator:

```
aliases      : o
               o
```

```
              o
            , register_discount_calculator
              for service_locatable~register_discount_calculator
```

4. Include the following empty method to class service_locator after the one for register_revenue_calculator:

```
method register_discount_calculator.
endmethod.
```

5. Include the following alias in the public section of class service_factory after the one for create_revenue_calculator:

```
  aliases      : o
                 o
                 o
               , create_discount_calculator
                 for service_creatable~create_discount_calculator
```

6. Include the following extra statement at the end of method create_all_service of class service_factory:

```
  me->create_discount_calculator( ).
```

7. Include the following empty method to class service_factory after the one for create_revenue_calculator:

```
method create_discount_calculator.
endmethod.
```

8. Add the following line to the clear statement in method clear_all_service_locators of class service_locator_test_helper:

```
  clear: o
         o
         o
       , service_locator=>singleton->discount_calculator
```

9. Include the following statement at the end of method clear_all_service_locators of class service_locator_test_helper:

```
  cl_abap_unit_assert=>assert_initial(
    act                       = service_locator=>singleton->discount_calculator
    ).
```

10. Include the following methods statement in the private secton of class service_factory_autester after the one for create_revenue_calculator:

```
  methods      : o
                 o
                 o
               , create_discount_calculator
                 for testing
```

11. Include the following statement at the end of method create_all_services of class service_factory_autester:

```
  cl_abap_unit_assert=>assert_not_initial(
    act                       = service_locator=>singleton->discount_calculator
    ).
```

12. Include the following statement at the end of method create_flights_organizer of class service_factory_autester:

```
  cl_abap_unit_assert=>assert_initial(
    act                       = service_locator=>singleton->discount_calculator
    ).
```

13. Include the following statement at the end of method create_flights_report of class service_factory_autester:

```
cl_abap_unit_assert=>assert_initial(
  act                       = service_locator=>singleton->discount_calculator
  ).
```

14. Include the following statement at the end of method create_revenue_calculator of class service_factory_autester:

```
cl_abap_unit_assert=>assert_initial(
  act                       = service_locator=>singleton->discount_calculator
  ).
```

15. Specify the following method implementation at the end of class service_factory_autester:

```
method create_discount_calculator.
  service_factory=>singleton->create_discount_calculator( ).
  cl_abap_unit_assert=>assert_not_bound(
    act                       = service_locator=>singleton->flights_organizer
    ).
  cl_abap_unit_assert=>assert_not_bound(
    act                       = service_locator=>singleton->flights_report
    ).
  cl_abap_unit_assert=>assert_not_bound(
    act                       = service_locator=>singleton->revenue_calculator
    ).
  cl_abap_unit_assert=>assert_not_initial(
    act                       = service_locator=>singleton->discount_calculator
    ).
endmethod.
```

Run

Action: Specify Airline ‘AA’, **discount 10**, ALV classic list and press Execute.

Result: Program fails with exception CX_SY_DYN_CALL_ILLEGAL_FUNC.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers error during methods create_all_services and create_discount_calculator; Status message indicates Processed: 1 program, 2 test classes, 14 test methods.

Remarks

With this version we have provided class service_locator with a method to register a flight discount calculator and provided class service_factory with a method to create and register a flight discount calculator. Both method implementations have been left empty for now.

We’ve also changed class service_locator_test_helper to clear and assert so on this new attribute of class service_locator. We’ve also adjusted all the unit test methods of class service_factory_autester to include an assertion of bound or not bound, as applicable, for this new service, and we’ve provided a new unit test method – create_discount_calculator – to assert that a service is provided after calling method create_discount_calculator of class service_factory.

The program still fails when run in production mode.

The unit test create_discount_calculator fails because even though it calls method create_discount_calculator of class service_factory, that method has no implementation.

19.3 Exercise 124

Program: ZAUT502C

Requirements

Reason for change

- Fix the unit test failures encountered in the previous version.

Changes to be applied

1. Specify the following implementation for method create_discount_calculator of class service_locator:

```
me->discount_calculator       = discount_calculator.
```

2. Specify the following implementation for method register_discount_calculator of class service_factory:

```
constants    : discount_calculator
                              type funcname  value 'ZCALCULATE_DISCOUNTED_AIRFARE'
             .
service_locator=>singleton->register_discount_calculator(
  exporting
    discount_calculator       = discount_calculator
    ).
```

Run

Action: Specify Airline ‘AA’, **discount 10**, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 14 test methods.

Remarks

With this version we have provided implementations for the empty methods register_discount_calculator of class service_locator and create_discount_calculator of class service_factory.

The program no longer fails when run in production mode and once again all unit tests pass.

19.4 Exercise 125

Program: ZAUT502D

Requirements

Reason for change

- Expand test coverage of subroutine adjust_flight_revenue.

Changes to be applied

1. In class tester, rename test method apply_flight_discount to apply_flight_discount_over_100.
2. After test method apply_flight_discount_over_100, define test method apply_flight_discount_50:
   - Add method definition for apply_flight_discount_50 to the private section of class tester following the definition for method apply_flight_discount_over_100:

```
methods      : o
               o
               o
             , apply_flight_discount_over_100
                 for testing
             , apply_flight_discount_50
                 for testing
```

   - Include the following method implementation after the implementation for method apply_flight_discount_over_100:

```
method apply_flight_discount_50.
  constants    : discount_50_percent
                               type num03     value 50
               .
  data         : flights_entry  like line
                                 of service_locator=>singleton->flights_organizer->flights_stack
               , flights_stack_before
                               type flights_organizable=>flights_list
               .
  cl_abap_unit_assert=>assert_not_initial(
    act                        = service_locator=>singleton->flights_organizer->flights_stack
    msg                        = 'No records available for testing flight discount'
    ).
  loop at service_locator=>singleton->flights_organizer->flights_stack
     into flights_entry.
    append flights_entry
        to flights_stack_before.
  endloop.
  perform apply_flight_discount using discount_50_percent
                            changing service_locator=>singleton->flights_organizer->flights_stack.
  cl_abap_unit_assert=>assert_equals(
    act                        = service_locator=>singleton->flights_organizer->flights_stack
    exp                        = flights_stack_before
    msg                        = 'Unequal discounted flights stacks'
    ).
endmethod.
```

Run

Action: Specify Airline ‘AA’, **discount 10**, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers error during method adjust flight_discount_50; Status message indicates Processed: 1 program, 2 test classes, 15 test methods.

Remarks

As we noted when registering issue #41, unit test apply_flight_discount does not call subroutine apply_flight_discount with calling parameters that would cause the flight discount calculation service to be invoked.

With this version we have renamed unit test method adjust_flight_discount of class tester to adjust_flight_discount_over_100, because it is using a discount parameter value greater than 100 to call subroutine apply_flight_discount. We also have provided a new unit test specifically to call subroutine apply_flight_discount with calling parameters that will cause the flight discount calculation service to be invoked.

The new unit test method adjust_flight_discount_50 fails because the *before* image and the *after* image of the flights_stack attribute of class flights_organizer are unequal after calling subroutine apply_flight_discount with the specified discount.

Let's register in our issues list that this version resolves issue #41.

#	Identified	Resolved	Description
1	ZAUT101A		No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields

#	Identified	Resolved	Description
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A	ZAUT402C	First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A	ZAUT402C	Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A	ZAUT402C	Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B	ZAUT402C	Fourth singleton class is introduced: flights_report_test_double
35	ZAUT302D	ZAUT302N	Unit test method show_flights of class tester explicitly calls test double

#	Identified	Resolved	Description
36	ZAUT302J	ZAUT302M	Use of singleton classes result in interacting tests
37	ZAUT401A		Fifth singleton class is introduced: service_locator
38	ZAUT401A	ZAUT401G	Class service_locator defines externally changeable public attribute flights_organizer
39	ZAUT401D	ZAUT401G	Class service_locator defines externally changeable public attribute flights_report
40	ZAUT402A		Sixth singleton class is introduced: service_factory
41	**ZAUT502A**	**ZAUT502D**	**Method apply_flight_discount of class tester fails to cause subroutine apply_flight_discount to call the function module**

19.5 Exercise 126

Program: ZAUT502E

Requirements

Reason for change

- Fix the unit test failure encountered in the previous version.

Changes to be applied

1. Via SE37, copy funciton module ZCALCULATE_DISCOUNTED_AIRFARE to ZCALC_DISCOUNT_AIRFARE_TSTDBL:
   - Provide it with the following implementation:

```
constants    : test_double_discount
                        type s_price   value '123.45'
             .
discount_fare               = test_double_discount.
```

   - Remove the definition of class tester after the ENDFUNCTION statement.

   Activate all function module components.

2. Include the following constant in method apply_flight_discount_50 of class tester:

```
constants    : o
               o
               o
             , discount_calculator
                        type funcname  value 'ZCALC_DISCOUNT_AIRFARE_TSTDBL'
```

3. Include the following data definition in method apply_flight_discount_50 of class tester after the one for flights_stack_before:

```
data         : o
```

```
              o
              o
            , test_double_discount_fare
                           type s_price
```

4. Include the following statements in method apply_flight_discount_50 of class tester ahead of the loop statement:

```
service_locator=>singleton->register_discount_calculator(
  exporting
    discount_calculator       = discount_calculator
    ).
call function service_locator=>singleton->discount_calculator
  exporting
    full_fare                 = 00
    discount                  = 00
  importing
    discount_fare             = test_double_discount_fare
  exceptions
    others                    = 0
    .
```

5. Include the following statement in method apply_flight_discount_50 of class tester ahead of the append statement:

```
  flights_entry-price         = test_double_discount_fare.
```

Run

Action: Specify Airline ‘AA’, **discount 10**, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 15 test methods.

Remarks

Here we have copied function module ZCALCULATE_DISCOUNTED_AIRFARE to ZCALC_DISCOUNT_AIRFARE_TSTDBL (test double) and changed its implementation to always return the same value as a discount, regardless of the values received through its importing parameters.

With this version we’ve changed unit test adjust_flight_discount_50 to (a) register function module ZCALC_DISCOUNT_AIRFARE_TSTDBL to the service locator as the entity to provide the flight discount service, then (b) call this service once to determine the discount value it always will return, and finally (c) use this value as the expected discount value for the records in attribute flights_stack of class flights_organizer after subroutine apply_flights_discount is called.

Once again, all unit tests pass.

20 ABAP Unit Testing 503 – Gaining Control Over Message Statements

This section describes the requirements for the exercise programs associated with the Chapter 10 section titled Using the Service Locator to Manage MESSAGE Statements in the book Automated Unit Testing with ABAP.

20.1 Exercise 127

Program: ZAUT503A

Requirements

Reason for change

- Change service locator to handle a message dispatcher.

Changes to be applied

1. Include the following interface ahead of interface flights_organizer_testable:

```
interface message_dispatchable.
    types        : message_type   type symsgty
                 , message_id     type symsgid
                 , message_number type symsgno
                 , message_text   type symsgv
                 .
    constants    : status_message type message_dispatchable=>message_type
                                          value 'S'
                 , information_message
                                  type message_dispatchable=>message_type
                                          value 'I'
                 , warning_message
                                  type message_dispatchable=>message_type
                                          value 'W'
                 , error_message  type message_dispatchable=>message_type
                                          value 'E'
                 , abort_message  type message_dispatchable=>message_type
                                          value 'A'
                 , exit_message   type message_dispatchable=>message_type
                                          value 'X'
                 .
    methods      : issue_identified_message
                     importing
                       message_severity
                         type message_dispatchable=>message_type
                           default message_dispatchable=>status_message
                       message_display_severity
                         type message_dispatchable=>message_type optional
                       id
                         type message_dispatchable=>message_id
                       number
                         type message_dispatchable=>message_number
                       text_01
                         type message_dispatchable=>message_text
                       text_02
                         type message_dispatchable=>message_text optional
                       text_03
                         type message_dispatchable=>message_text optional
                       text_04
                         type message_dispatchable=>message_text optional
                 , issue_unidentified_message
                     importing
                       message_severity
                         type message_dispatchable=>message_type
                           default message_dispatchable=>status_message
                       message_display_severity
                         type message_dispatchable=>message_type optional
                       text
                         type clike
```

```
                .
endinterface.
```

2. Include the following attribute in the public section of class service_locator after the one for discount_calculator:

```
data        : o
              o
              o
            , message_dispatcher
                          type ref
                            to message_dispatchable
                                 read-only
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 15 test methods.

Remarks

Since we already know (via issues #18, #19 and #20) that certain message severity levels used with the ABAP MESSAGE statement interferes with the ability of a unit test to run to completion, we have undertaken to implement a message service capability to be made available via the service locator.

With this version we have defined a new interface called message_dispatchable and added to the service locator a new public attribute in which to register the entity to provide a message dispatch service. With this new attribute it is now capable of managing 5 services. Notice that interface message_dispatchable defines two methods:

- Method issue_identified_message is intended to accommodate all the parameters that could be specified when using either of the following formats of the ABAP MESSAGE statement:

```
message t(nnn) …
message id … type … number … …
```

- Method issue_unidentified_message is intended to accommodate the text, message type and display parameters that could be specified when using the following format of the ABAP MESSAGE statement:

```
message text type mtype …
```

Both method definitions are intended to be implemented by classes that can provide corresponding method implementations such that a call to the method can replace a corresponding ABAP MESSAGE statement. Notice that neither of these method definitions is equipped to handle the “into” clause of a message statement, since the presence of that clause would have the effect of causing no message to be issued during execution.

20.2 Exercise 128

Program: ZAUT503B

Requirements

Reason for change

- Change classes service_locator_test_helper and service_factory_autester to include message dispatcher service.

Changes to be applied

1. In method clear_all_service_locators of class service_locator_test_helper, do the following:
   - Add the following line to the clear statement:

```
clear: o
       o
       o
     , service_locator=>singleton->message_dispatcher
```

   - Add the following assertion at the end of the method:

```
cl_abap_unit_assert=>assert_not_bound(
  act                        = service_locator=>singleton->message_dispatcher
  ).
```

2. Add the following test method definition to class service_factory_autester after the one for create_discount_calculator:

```
methods      : o
               o
               o
             , create_message_dispatcher
                 for testing
```

3. Add the following assertion to the end of method create_all_services of class service_factory_autester:

```
cl_abap_unit_assert=>assert_bound(
  act                        = service_locator=>singleton->message_dispatcher
  ).
```

4. Add the following assertion to the end of method create_flights_organizer of class service_factory_autester:

```
cl_abap_unit_assert=>assert_not_bound(
  act                        = service_locator=>singleton->message_dispatcher
  ).
```

5. Add the following assertion to the end of method ccreate_flights_report of class service_factory_autester:

```
cl_abap_unit_assert=>assert_not_bound(
  act                        = service_locator=>singleton->message_dispatcher
  ).
```

6. Add the following assertion to the end of method create_revenue_calculator of class service_factory_autester:

```
cl_abap_unit_assert=>assert_not_bound(
  act                        = service_locator=>singleton->message_dispatcher
  ).
```

7. Add the following assertion to the end of method create_discount_calculator of class service_factory_autester:

```
cl_abap_unit_assert=>assert_not_bound(
  act                        = service_locator=>singleton->message_dispatcher
```

```
    ).
```

8. Add the following method implementation at the end of the class:

```
method create_message_dispatcher.
  cl_abap_unit_assert=>assert_not_bound(
    act                         = service_locator=>singleton->flights_organizer
    ).
  cl_abap_unit_assert=>assert_not_bound(
    act                         = service_locator=>singleton->flights_report
    ).
  cl_abap_unit_assert=>assert_not_bound(
    act                         = service_locator=>singleton->revenue_calculator
    ).
  cl_abap_unit_assert=>assert_initial(
    act                         = service_locator=>singleton->discount_calculator
    ).
  cl_abap_unit_assert=>assert_bound(
    act                         = service_locator=>singleton->message_dispatcher
    ).
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers error during methods create_all_services and create_message_dispatcher Status message indicates Processed: 1 program, 2 test classes, 16 test methods.

Remarks

With this version we’ve changed the service_locator_test_helper to accommodate the new message dispatcher public attribute and changed the unit test methods of class service_factory_autester to include this attribute in their respective assertions, as well as adding a new unit test – create_message_dispatcher – to service_factory_autester to assert that a message dispatcher entity can be registered to the service locator.

The new unit test fails because neither the service locator nor the service factory have been updated to accommodate creating and registering a message dispatch service.

20.3 Exercise 129

Program: ZAUT503C

Requirements

Reason for change

- Fix the unit test failures encountered in the previous version.

Changes to be applied

1. Add the following method definition to interface service_locatable after the one for register_discount_calculator:

```
    methods      : o
                   o
                   o
                 , register_message_dispatcher
                     importing
                       message_dispatcher
                         type ref
                           to message_dispatchable
```

2. Add the following method definition to interface service_creatable after the one for create_discount_calculator:

```
    methods      : o
                   o
                   o
                 , create_message_dispatcher
```

3. Add the following to the aliases statement of class service_locator after the one for register_discount_calculator:

```
    aliases      : o
                   o
                   o
                 , register_message_dispatcher
                     for service_locatable~register_message_dispatcher
```

4. Add the following method implementation to class service_locator after the one for register_discount_calculator:

```
  method register_message_dispatcher.
    me->message_dispatcher        = message_dispatcher.
  endmethod.
```

5. Add the following new class definition after class service_locator:

```
class messenger                       definition
                                      final
                                      .
  public section.
    interfaces   : message_dispatchable
                 .
    aliases      : issue_identified_message
                     for message_dispatchable~issue_identified_message
                 , issue_unidentified_message
                     for message_dispatchable~issue_unidentified_message
                 .
endclass.
class messenger                       implementation.
  method issue_identified_message.
    data         : message_display_type
                                  type message_dispatchable=>message_type
                 .
    message_display_type          = message_display_severity.
    if message_display_type is initial.
      message_display_type        = message_severity.
    endif.
    " Issue message using message statement:
    message id      id
            type    message_severity
            number  number
            display like message_display_type
            with    text_01
                    text_02
                    text_03
                    text_04
            .
  endmethod.
  method issue_unidentified_message.
```

```
    data         : message_display_type
                                type message_dispatchable=>message_type
                 .
    message_display_type          = message_display_severity.
    if message_display_type is initial.
      message_display_type        = message_severity.
    endif.
    " Issue message using message statement:
    message text type message_severity display like message_display_type.
  endmethod.
endclass.
```

6. Add the following to the aliases statement of class service_factory after the one for create_discount_calculator:

```
    aliases      : o
                   o
                   o
                 , create_message_dispatcher
                     for service_creatable~create_message_dispatcher
```

7. Add the following statement to the end of method create_all_services of class service_factory:

```
    me->create_message_dispatcher( ).
```

8. Add the following method implementation to class service_factory after the one for create_discount_calculator:

```
  method create_message_dispatcher.
    data         : message_dispatcher
                                type ref
                                  to message_dispatchable
                 .
    create object message_dispatcher
             type messenger.
    service_locator=>singleton->register_message_dispatcher(
      exporting
        message_dispatcher        = message_dispatcher
        ).
  endmethod.
```

9. Add the following statement to the start of method create_message_dispatcher of class service_factory_autester:

```
    service_factory=>singleton->create_message_dispatcher( ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 16 test methods.

Remarks

With this version we have provided both the service locator and the service factory with public methods to accommodate creating and registering a message dispatch service. This includes creating a new class – messenger – that implements the message_dispatchable interface and through class service_factory is created

and registered as the message dispatch service. Also, now that there is a method create_message_dispatcher defined and implemented by class service_factory, unit test method create_message_dispatcher of class service_factory_autester now calls it before performing its assertions.

Notice that the implementations specified for the methods of class messenger simply cause them to issue their respective ABAP MESSAGE statements.

Once again, all unit tests pass.

20.4 Exercise 130

Program: ZAUT503D

Requirements

Reason for change

- Change subroutine show_flights_count to use the message dispatch service in place of the ABAP MESSAGE statement.

Changes to be applied

1. In subroutine show_flights_count, replace the message statement with the following:

```
service_locator=>singleton->message_dispatcher->issue_identified_message(
  exporting
    id                      = '0k'
    number                  = 000
    text_01                 = conv #( flights_count )
    text_02                 = 'flights are available for carrier'
    text_03                 = conv #( carrier )
  ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria, but status message is preceded by "0k:000" and many spaces separate this string from the message text.

Note: Corresponding online help for ABAP statement MESSAGE indicates the following:

> If the specified message is not found for the logon language of the current user, a search is made in the secondary language (profile parameter zcsa/second_language) and then in English. If it is still not found, the specified message type, message class, and message number are used as short text in uppercase letters and separated by a colon ":".

This appears to be occurring because we specified the message class as "0k", using a lower case "k".

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 16 test methods.

Remarks

With this version we have replaced the ABAP MESSAGE statement in subroutine show_flights_count with a call to our new message dispatcher service. Although it works correctly, we are seeing some unexpected formatting of the status message that appears on the screen when the program is run in production mode.

Notice that this is the first time we are using what is described by SAP as a Constructor Operator, new additions to the ABAP syntax with release 7.40, SP2. In this case we are using the CONV operator to indicate that the value in parenthesis following it is to be converted into a value of a different type, the type indication preceding the parenthesis, or, when the type indication is "#', as we have used here, to determine the type dynamically by using the type of the receiving field. Accordingly, for these parameters of our statement to call to method issue_identified_message of the instance of service_locator=>singleton->message_dispatcher:

```
text_01                  = conv #( flights_count )
text_03                  = conv #( carrier )
```

the values flights_count (defined as type int4) and carrier (defined as type s_carr_id) are converted into the same type of data as parameters text_01 (type symsgv) and text_03 (also type symsgv), respectively.

Note: If you are using an SAP environment where the ABAP compiler does not yet recognize the new Constructor Operators, then simply define intermediate fields into which these values can be moved, defined using type symsgv, move these values into those fields prior to making this method call and change the method call to use the corresponding intermediate fields instead.

20.5 Exercise 131

Program: ZAUT503E

Requirements

Reason for change

- Eliminate odd formatting of status message presented in previous version.

Changes to be applied

1. In subroutine show_flights_count, specify upper case "K" for the id parameter.

Run

Action: Specify Airline 'AA', no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria, and neither is message preceded by "0k:000" nor are there many spaces separate this string from the message text.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 16 test methods.

Remarks

With this version we simply have changed the lowercase “k” in message class “0k” to an uppercase value. Now when run in production mode the status message appearing on the screen is formatted as we expect.

20.6 Exercise 132

Program: ZAUT503F

Requirements

Reason for change

- Enable testing the code in classic event block at selection-screen.

Changes to be applied

1. Define subroutine process_selection using the following model, placing it ahead of subroutine present_report:

```
form process_selection using discount
                             type discount
                           carrier
                             type flights_organizable=>carrier.
endform.
```

2. Move to it all code in the at selection-screen event block after the sy-ucomm check.
3. Replace the code moved from the at selection-screen event block with the following statement:

```
perform process_selection using discount
                                carrier.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 16 test methods.

Remarks

There are other ABAP MESSAGE statements in the program that we’d like to be able to cover with unit tests, but these appear within the scope of the “at selection-screen” classic event block. With this version we have refactored the program so that the logic in the “at selection-screen” classic event block is relocated to new subroutine process_selection.

The unit tests still pass, so yet again we have confirmation that the refactoring we applied to the the program caused none of the tests to fail.

Let’s register in our issues list that this version resolves issue #1.

#	Identified	Resolved	Description
1	**ZAUT101A**	**ZAUT503F**	**No test for code in classic event block at-selection screen**
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F		Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G		Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H		Unit test issues Runtime Error upon encountering MESSAGE statement with severity X

#	Identified	Resolved	Description
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A	ZAUT402C	First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A	ZAUT402C	Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A	ZAUT402C	Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B	ZAUT402C	Fourth singleton class is introduced: flights_report_test_double
35	ZAUT302D	ZAUT302N	Unit test method show_flights of class tester explicitly calls test double
36	ZAUT302J	ZAUT302M	Use of singleton classes result in interacting tests
37	ZAUT401A		Fifth singleton class is introduced: service_locator
38	ZAUT401A	ZAUT401G	Class service_locator defines externally changeable public attribute flights_organizer
39	ZAUT401D	ZAUT401G	Class service_locator defines externally changeable public attribute flights_report
40	ZAUT402A		Sixth singleton class is introduced: service_factory

#	Identified	Resolved	Description
41	ZAUT502A	ZAUT502D	Method apply_flight_discount of class tester fails to cause subroutine apply_flight_discount to call the function module

20.7 Exercise 133

Program: ZAUT503G

Requirements

Reason for change

- Change class tester to provide more combinations of calling parameters for testing subroutine process_selection.

Changes to be applied

1. In class tester, define the following methods for testing, after the definition for method teardown:
   - process_selection_bad_discount
   - process_selection_bad_carrier
   - process_selection_good_carrier

   The implementation for each one should contain the following test assertion:

```
cl_abap_unit_assert=>fail(
  msg                        = 'Test method not implemented'
  ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers error during methods process_selection_bad_discount, process_selection_bad_carrier and process_selection_good_carrier; Status message indicates Processed: 1 program, 2 test classes, 19 test methods.

Remarks

With this version we have defined the following new unit test methods to class tester:

- process_selection_bad_discount
- process_selection_bad_carrier
- process_selection_good_carrier

Each one calls the new subroutine introduced with the previous version. Because there is conditional logic in new subroutine process_selection (the same conditional logic that existed already in the classic event block from

which it was copied) we have defined 3 new unit tests to accommodate (a) the path required to reach the warning message statement regarding the discount, (b) the path required to reach the error message statement regarding the carrier and (c) the path required to cause both message statements to be bypassed.

These new unit tests all fail because all of them include only a call to method fail of class cl_abap_unit_assert.

20.8 Exercise 134

Program: ZAUT503H

Requirements

Reason for change

- Fix one of the 3 failing methods of class tester.

Changes to be applied

1. In class tester, replace the test assertion implemented for method process_selection_good_carrier with the following implementation:

```
set_bogus_message( ).
assert_message_is_bogus( ).
perform process_selection using 00
                         american_airlines.
" With these specified calling parameters, no message should be issued by
" subroutine process_selection, so the message content should remain unchanged:
assert_message_is_bogus( ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers error during methods process_selection_bad_discount and process_selection_bad_carrier; Status message indicates Processed: 1 program, 2 test classes, 19 test methods.

Remarks

With this version we have resolved only one of the 3 failing unit tests by providing it with an implementation to assert the results we should expect from it.

Two unit tests still are failing.

20.9 Exercise 135

Program: ZAUT503I

Requirements

Reason for change

- Fix another one of the 3 failing methods of class tester.

Changes to be applied

1. In class tester, replace the test assertion implemented for method process_selection_bad_discount with the following implementation:

```
set_bogus_message( ).
assert_message_is_bogus( ).
perform process_selection using 110
                         american_airlines.
" With these specified calling parameters, a warning message should be issued by
" subroutine process_selection, so the message content should be changed:
assert_message_not_bogus( ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces ALV classic list display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers error during method process_selection_bad_carrier; Status message indicates Processed: 1 program, 2 test classes, 19 test methods.

Remarks

With this version we have resolved another one of the failing unit tests by providing it with an implementation to assert the results we should expect from it.

One unit test still is failing.

20.10 Exercise 136

Program: ZAUT503J

Requirements

Reason for change

- Fix the final one of the 3 failing methods of class tester.

Changes to be applied

1. In class tester, replace the test assertion implemented for method process_selection_bad_carrier with the following implementation:

```
    constants    : bogus_carrier  type flights_organizable=>carrier
                                          value '??'
                 .
    set_bogus_message( ).
    assert_message_is_bogus( ).
    perform process_selection using 00
                            bogus_carrier.
    " With these specified calling parameters, an error message should be issued by
    " subroutine process_selection, so the message content should be changed:
    assert_message_not_bogus( ).
```

Run

Action: Specify **Airline '??'**, no discount, ALV classic list and press Execute.

Result: Message appears at bottom of screen indicating "No flights match carrier ??".

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester still triggers error during method process_selection_bad_carrier: Exception Error <CX_AUNIT_UNCAUGHT_MESSAGE>; Status message indicates Processed: 1 program, 2 test classes, 19 test methods.

Remarks

With this version we have provided the remaining failing unit test with an implementation to assert the results we should expect from it.

This unit test – process_selection_bad_carrier – still fails because it calls subroutine process_selection using parameter values which cause it to encounter the error message statement.

20.11 Exercise 137

Program: ZAUT503K

Requirements

Reason for change

- Introduce test double for class messenger.

Changes to be applied

1. Define class messenger_test_double as follows, placing it after class messenger:

```
class messenger_test_double           definition
                                      final
                                      for testing
                                      .
  public section.
    interfaces   : message_dispatchable
                 .
    aliases      : issue_identified_message
                     for message_dispatchable~issue_identified_message
                 , issue_unidentified_message
                     for message_dispatchable~issue_unidentified_message
                 .
    types        : begin of identified_message_row
```

```
                ,   type         type message_dispatchable=>message_type
                ,   display_type type message_dispatchable=>message_type
                ,   id           type message_dispatchable=>message_id
                ,   number       type message_dispatchable=>message_number
                ,   text_01      type message_dispatchable=>message_text
                ,   text_02      type message_dispatchable=>message_text
                ,   text_03      type message_dispatchable=>message_text
                ,   text_04      type message_dispatchable=>message_text
                , end   of identified_message_row
                , identified_message_list
                                 type standard table
                                   of identified_message_row
                , begin of unidentified_message_row
                ,   type         type message_dispatchable=>message_type
                ,   display_type type message_dispatchable=>message_type
                ,   text         type message_dispatchable=>message_text
                , end   of unidentified_message_row
                , unidentified_message_list
                                 type standard table
                                   of unidentified_message_row
                .
    data        : identified_message_stack
                                 type identified_message_list
                                        read-only
                , unidentified_message_stack
                                 type unidentified_message_list
                                        read-only
                .
endclass.
class messenger_test_double           implementation.
  method issue_identified_message.
    data        : identified_message_entry
                                 like line
                                   of identified_message_stack
                .
    identified_message_entry-type
                                 = message_severity.
    if message_display_severity is not initial.
      identified_message_entry-display_type
                                 = message_display_severity.
    else.
      identified_message_entry-display_type
                                 = message_severity.
    endif.
    identified_message_entry-id   = id.
    identified_message_entry-number
                                 = number.
    identified_message_entry-text_01
                                 = text_01.
    identified_message_entry-text_02
                                 = text_02.
    identified_message_entry-text_03
                                 = text_03.
    identified_message_entry-text_04
                                 = text_04.
    append identified_message_entry
        to identified_message_stack.
    sy-msgty                     = identified_message_entry-type.
    sy-msgid                     = identified_message_entry-id.
    sy-msgno                     = identified_message_entry-number.
    sy-msgv1                     = identified_message_entry-text_01.
    sy-msgv2                     = identified_message_entry-text_02.
    sy-msgv3                     = identified_message_entry-text_03.
    sy-msgv4                     = identified_message_entry-text_04.
  endmethod.
  method issue_unidentified_message.
    constants    : unidentified_message_id
                                 type message_dispatchable=>message_id
                                        value '00'
                , unidentified_message_number
                                 type message_dispatchable=>message_number
                                        value '000'
                .
    data        : unidentified_message_entry
                                 like line
                                   of unidentified_message_stack
                , begin of message_content
                ,   text_01      type message_dispatchable=>message_text
                ,   text_02      type message_dispatchable=>message_text
```

```
                ,   text_03      type message_dispatchable=>message_text
                ,   text_04      type message_dispatchable=>message_text
                , end   of message_content
                .
    unidentified_message_entry-type
                                 = message_severity.
    if message_display_severity is not initial.
      unidentified_message_entry-display_type
                                 = message_display_severity.
    else.
      unidentified_message_entry-display_type
                                 = message_severity.
    endif.
    unidentified_message_entry-text
                                 = text.
    append unidentified_message_entry
        to unidentified_message_stack.
    sy-msgty                     = unidentified_message_entry-type.
    sy-msgid                     = unidentified_message_id.
    sy-msgno                     = unidentified_message_number.
    message_content              = text.
    sy-msgv1                     = message_content-text_01.
    sy-msgv2                     = message_content-text_02.
    sy-msgv3                     = message_content-text_03.
    sy-msgv4                     = message_content-text_04.
  endmethod.
endclass.
```

Run

Action: Specify **Airline '??'**, no discount, ALV classic list and press Execute.

Result: Message appears at bottom of screen indicating "No flights match carrier ??".

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester still triggers error during method process_selection_bad_carrier: Exception Error <CX_AUNIT_UNCAUGHT_MESSAGE>; Status message indicates Processed: 1 program, 2 test classes, 19 test methods.

Remarks

With this version we have defined new class messenger_test_double, which, like class messenger, implements the message_dispatchable interface. Notice that the implementation of its methods causes it to record calls made to them instead of issuing the corresponding message statement as is done by class messenger.

Because it is capable of recording the calls made to it, this new test double class represents what is known as a test spy – a test double that is capable of recording information about its use for later reference (see xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 137).

Unit test process_selection_bad_carrier still fails because we have not made any changes to cause it to pass … yet.

20.12 Exercise 138

Program: ZAUT503L

Requirements

Reason for change

- Fix the unit test failure encountered in the previous version.

Changes to be applied

1. In subroutine process_selection, replace the message statement after the call to method service_locator=>singleton->flights_organizer->get_flights_count with the following:

```
service_locator=>singleton->message_dispatcher->issue_identified_message(
  exporting
    message_severity        = message_dispatchable=>error_message
    id                      = '0K'
    number                  = 000
    text_01                 = 'No flights match carrier'
    text_03                 = conv #( carrier )
  ).
```

2. In method setup of class tester:
   - Define data field messenger_test_double as reference to message_dispatchable following flights_report_test_double:

```
data         : o
               o
               o
             , messenger_test_double
                             type ref
                               to message_dispatchable
```

   - Add the following code to the end of the method:

```
" Instantiate messenger_test_double:
create object messenger_test_double
        type messenger_test_double.
" Register messenger_test_double as message_dispatcher service:
service_locator=>singleton->register_message_dispatcher(
  exporting
    message_dispatcher        = messenger_test_double
  ).
```

Run

Action: Specify **Airline '??'**, no discount, ALV classic list and press Execute.

Result: Message appears at bottom of screen indicating "No flights match carrier ??".

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 19 test methods.

Remarks

With this version we have changed subroutine process_selection to replace its error message statement with a call to the messenger service provided by the service locator. Also, method setup of class tester was changed to create an instance of class messenger_test_double and register it with the service locator as the messenger service.

Unit test method process_selection_bad_carrier now passes and running the program in production mode as described above still causes an error message to appear.

Let's register in our issues list that this version resolves issue #18, and by implication also resolves issues #19 and #20.

#	Identified	Resolved	Description
1	ZAUT101A	ZAUT503F	No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	**ZAUT105F**	**ZAUT503L**	**Unit test issues Exception Error upon encountering MESSAGE statement with severity E**
19	**ZAUT105G**	**(ZAUT503L)**	**Unit test issues Exception Error upon encountering MESSAGE statement with severity A**

#	Identified	Resolved	Description
20	**ZAUT105H**	**(ZAUT503L)**	**Unit test issues Runtime Error upon encountering MESSAGE statement with severity X**
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N		Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A	ZAUT402C	First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A	ZAUT402C	Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A	ZAUT402C	Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B	ZAUT402C	Fourth singleton class is introduced: flights_report_test_double
35	ZAUT302D	ZAUT302N	Unit test method show_flights of class tester explicitly calls test double
36	ZAUT302J	ZAUT302M	Use of singleton classes result in interacting tests
37	ZAUT401A		Fifth singleton class is introduced: service_locator
38	ZAUT401A	ZAUT401G	Class service_locator defines externally changeable public attribute flights_organizer
39	ZAUT401D	ZAUT401G	Class service_locator defines externally changeable public attribute flights_report

#	Identified	Resolved	Description
40	ZAUT402A		Sixth singleton class is introduced: service_factory
41	ZAUT502A	ZAUT502D	Method apply_flight_discount of class tester fails to cause subroutine apply_flight_discount to call the function module

20.13 Exercise 139

Program: ZAUT503M

Requirements

Reason for change

- Provide more test coverage to subroutine process_selection.

Changes to be applied

1. Add new test method process_selection_bad_values to class tester using the following implementation:
   - Add method definition for process_selection_bad_values to the private section of class tester after the definition for method teardown:

```
, process_selection_bad_values
    for testing
```

   - Include the following method implementation after the implementation for method teardown:

```
method process_selection_bad_values.
  constants    : bogus_carrier  type flights_organizable=>carrier
                                      value '??'
               .
  data         : messenger_test_double
                              type ref
                                to messenger_test_double
               .
  set_bogus_message( ).
  assert_message_is_bogus( ).
  perform process_selection using 110
                           bogus_carrier.
  " With these specified calling parameters, both a warning message and
  " an error message should be issued by subroutine process_selection,
  " so the message content should be changed:
  assert_message_not_bogus( ).
  " We also should find that there are two identified messages registered in the
  " messenger_test_double; one is a warning message and the other is an error message:
  try.
    messenger_test_double       ?= service_locator=>singleton->message_dispatcher.
  catch cx_sy_move_cast_error.
    cl_abap_unit_assert=>fail(
      msg                       = 'Caught exception in test method process_selection_bad_values'
     ).
  endtry.
  cl_abap_unit_assert=>assert_equals(
    act                         = lines( messenger_test_double->identified_message_stack )
    exp                         = 02
    msg                         = 'Unexpected number of messages held by messenger test double'
    ).
endmethod.
```

Run

Action: Specify **Airline '??'**, **discount 110**, ALV classic list and press Execute.

Result: First a warning message appears at bottom of screen indicating "Fare discount percentage exceeding 100 will be ignored", then after pressing enter an error message appears at bottom of screen indicating "No flights match carrier ??".

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers error during method process_selection_bad_values; Status message indicates Processed: 1 program, 2 test classes, 20 test methods.

Remarks

With this version we have added a new unit test method – process_selection_bad_values – to class tester to test yet another call to subroutine process_selection, this time using calling parameters that will cause both the warning message and the error message to be issued.

The unit test fails because subroutine process_selection still has an ABAP MESSAGE statement issuing the warning message, and the unit test is expecting that both the warning message and the error message are being issued by a call to the messenger service.

Recall that test double class messenger_test_double, introduced with version ZAUT503K, represents a test spy. Here we are accessing this test spy to perform the "Behavior Verification" (xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 468) confirming that calls to its methods have recorded the number of messages that we expect it should have recorded.

20.14 Exercise 140

Program: ZAUT503N

Requirements

Reason for change

- Fix the unit test failure encountered in the previous version.

Changes to be applied

1. In subroutine process_selection, replace the warning message statement checking for discount greater than 100 with the following statement:

```
service_locator=>singleton->message_dispatcher->issue_identified_message(
  exporting
    message_severity        = message_dispatchable=>warning_message
    id                      = '0K'
    number                  = 000
    text_01                 = 'Fare discount percentage exceeding 100'
    text_03                 = 'will be ignored'
  ).
```

Run

Action: Specify **Airline '??'**, **discount 110**, ALV classic list and press Execute.

Result: First a warning message appears at bottom of screen indicating "Fare discount percentage exceeding 100 will be ignored", then after pressing enter an error message appears at bottom of screen indicating "No flights match carrier ??".

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 20 test methods

Remarks

With this version we have changed subroutine process_selection to replace its warming message statement with a call to the messenger service provided by the service locator.

Once again, all unit tests pass and running the program in production mode still causes both a warning message followed by an error message to appear. We have effectively gained control over the ABAP MESSAGE statements during the unit test run.

20.15 Exercise 141

Program: ZAUT503O

Requirements

Reason for change

- Assert number and type of messages issued during test of subroutine process_selection.

Changes to be applied

1. In method process_selection_bad_values of class tester, do the following:
   - Include the following data definitions on the data statement:

```
data        : o
              o
              o
            , identified_message_entry
                            like line
                              of messenger_test_double->identified_message_stack
            , error_message_count
                            type int4
            , warning_message_count
                            type int4
```

   - Include the following code at the end of the method:

```
loop at messenger_test_double->identified_message_stack
   into                        identified_message_entry.
  case identified_message_entry-type.
    when message_dispatchable=>error_message.
      add 01 to error_message_count.
    when message_dispatchable=>warning_message.
      add 01 to error_message_count.
  endcase.
endloop.
cl_abap_unit_assert=>assert_equals(
  act                         = error_message_count
  exp                         = 01
  msg                         = 'Unexpected number of error messages held by messenger test double'
```

```
  ).
cl_abap_unit_assert=>assert_equals(
  act                          = warning_message_count
  exp                          = 01
  msg                          = 'Unexpected number of warning messages held by messenger test double'
  ).
```

Run

Action: Specify **Airline '??'**, **discount 110**, ALV classic list and press Execute.

Result: First a warning message appears at bottom of screen indicating "Fare discount percentage exceeding 100 will be ignored", then after pressing enter an error message appears at bottom of screen indicating "No flights match carrier ??".

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester again triggers error during method process_selection_bad_values; Status message indicates Processed: 1 program, 2 test classes, 20 test methods.

Remarks

With this version we have included additional assertions in unit test process_selection_bad_values of class tester. Specifically, we want it to assert that the messenger test double was called once with a warning message and once again with an error message.

This unit test now fails due to a bug we introduced with the additional unit test code.

20.16 Exercise 142

Program: ZAUT503P

Requirements

Reason for change

- Fix the bug introduced in the previous version

Changes to be applied

1. In method process_selection_bad_values of class tester, change the statement after the statement ...

```
when message_dispatchable=>warning_message.
```

… from ...

```
add 01 to error_message_count.
```

... to:

```
add 01 to warning_message_count.
```

Run

Action: Specify **Airline '??'**, **discount 110**, ALV classic list and press Execute.

Result: First a warning message appears at bottom of screen indicating "Fare discount percentage exceeding 100 will be ignored", then after pressing enter an error message appears at bottom of screen indicating "No flights match carrier ??".

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 20 test methods.

Remarks

With this version we corrected the bug we had introduced with the code added to the unit test with the previous version – specifically, that when a message is of type warning it should increment the warning message counter and not the error message counter.

Once again, all unit tests pass. This means that unit test process_selection_bad_values has determined that not only has the messenger test double been invoked twice, but that one of those was for issuing a warning message and the other was for issuing an error message, just as we would expect to occur when we send both a bad discount value and a bad carrier value on the call to subroutine process_selection.

21 ABAP Unit Testing 504 – Gaining Control Over List Processing Statements

This section describes the requirements for the exercise programs associated with the Chapter 10 section titled Using the Service Locator to Manage List Processing Statements in the book Automated Unit Testing with ABAP.

21.1 Exercise 143

Program: ZAUT504A

Requirements

Reason for change

- Create a class that produces a report using WRITE statements and change the service factory to create and register an instance of that class for the flights_report service.

Changes to be applied

1. Include the following class after class flights_report:

```
class flights_report_old_format        definition
                                       final
                                       friends flights_report_testable
                                       .
  public section.
    interfaces   : flights_reportable
                   .
    aliases      : show_flights
                       for flights_reportable~show_flights
                   .
endclass.
class flights_report_old_format        implementation.
  method show_flights.
    data         : flights_entry  like line
                                    of flights_stack
                   .
    loop at flights_stack
       into flights_entry.
      new-line.
      write: flights_entry-carrid
           , flights_entry-connid
           , flights_entry-fldate
           , flights_entry-price
           , flights_entry-currency
           , flights_entry-planetype
           , flights_entry-seatsmax
           , flights_entry-seatsocc
           , flights_entry-paymentsum
           , flights_entry-seatsmax_b
           , flights_entry-seatsocc_b
           , flights_entry-seatsmax_f
           , flights_entry-seatsocc_f
           .
    endloop.
  endmethod.
endclass.
```

2. Change method create_flights_report of class service_factory to create a flights_report object of type flights_report_old_format:

```
    create object flights_report
             type flights_report_old_format.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 20 test methods.

Remarks

With this version we have introduced a new class implementing the flights_reportable interface. Its intent is reflected in its name: flights_report_old_format. It produces the flights report using ABAP WRITE statements to create a classical list (old format). The use of classical lists is contrary to “Rule 5.20: Use the SAP List Viewer” defined in the book "Official ABAP Programming Guidelines" (Keller, Thummel, 2010, SAP Press).

In addition, we’ve changed method create_flights_report of class service_factory to provide an instance of this class as the flights report service.

Notice that running this program in production mode now produces a classical list of flights.

Note: Despite our intentional decision to define a class that uses a deprecated output format, notice how easy it was to create a class to provide this capability and, because it implements the flights_reportable interface, to register it as the flights report service to the service locator. This demonstrates the flexibility of program design that becomes available through the use of the object-oriented model.

21.2 Exercise 144

Program: ZAUT504B

Requirements

Reason for change

- Prevent method setup of class tester from overriding the flights_report service with a test double.

Changes to be applied

1. In method setup of class tester, comment out the call to service_locator=>singleton->register_flights_report.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Content of write statement appears on the screen but is preceded by red and green highlighted rows indicating the following:

```
+--------------------------------------------------------------------------------
|Internal Session for Isolated Test Class Execution
+--------------------------------------------------------------------------------
|Warning
|=======
|Program: <program name>
|Class:   <test class name>
|This window is displayed because your test case has triggered
|a list command like follows:
|- new page
|- leave to list processing
|- uline
|- write
|- new page
|- ...
|This as any interactive technique is not permitted !!
+--------------------------------------------------------------------------------
|Please avoid the use of these statements. To locate them in the source code
|setting break-points on the mentioned statements should help.
+--------------------------------------------------------------------------------
|       <content of write statement appears here>
```

Pressing Back, Exit, Cancel or ESCape results in ABAP Unit: Results Display report indicating test class tester triggers errors during methods present_report, set_alv_field_catalog, set_alv_function_module_name and show_flights; Status message indicates Processed: 1 program, 2 test classes, 20 test methods.

Remarks

With this version we have removed from method setup of class tester the code to register the flights_report_test_double as the flights report service. The result is that when the unit tests are run, any calls to utilize the flights report service will use this service that had been created and registered by the service_factory – which now is the one that produces a report using the classical list statements.

Notice that now when running the unit tests, progress is interrupted by the presentation of the “Internal Session for Isolated Test Class Execution” list, requiring user intervention to allow the unit tests to run to completion. We’ve seen this before and it is registered as issue #24 in our issues list.

Some unit tests of class tester fail because they were expecting to find a flights report service being provided by an instance of flights_report_test_double and now are finding that this service is being provided by an instance of flights_report_old_format, preventing them from performing their respective assertions.

21.3 Exercise 145

Program: ZAUT504C

Requirements

Reason for change

- Begin to make changes to prevent Internal Session for Isolated Test Class Execution report from appearing during test.

Changes to be applied

1. Add the following interface definition ahead of interface message_dispatchable:

```
interface report_writable.
    types        : value_type     type c length 100
                 , format_type    type c length 100
                 .
    methods      : new_line
                 , write
                     importing
                       format
                         type report_writable=>format_type
                       value
                         type report_writable=>value_type
                 .
endinterface.
```

2. In the public section of class service_locator, add the following attribute after the one for message_dispatcher:

```
    data         : o
                   o
                   o
                 , report_writer  type ref
                                    to report_writable
                                         read-only
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Content of write statement appears on the screen but is preceded by red and green highlighted rows indicating the following:

```
+------------------------------------------------------------------------------
|Internal Session for Isolated Test Class Execution
+------------------------------------------------------------------------------
|Warning
|=======
|Program: <program name>
|Class:   <test class name>
|This window is displayed because your test case has triggered
|a list command like follows:
|- new page
|- leave to list processing
|- uline
|- write
|- new page
|- ...
|This as any interactive technique is not permitted !!
+------------------------------------------------------------------------------
```

```
|Please avoid the use of these statements. To locate them in the source code
|setting break-points on the mentioned statements should help.
+--------------------------------------------------------------------------
|       <content of write statement appears here>
```

Pressing Back, Exit, Cancel or ESCape results in ABAP Unit: Results Display report indicating test class tester triggers errors during methods present_report, set_alv_field_catalog, set_alv_function_module_name and show_flights; Status message indicates Processed: 1 program, 2 test classes, 20 test methods.

Remarks

With this version we have defined a new interface report_writable and added a new public attribute to the service locator to manage a report writer service. With this new attribute it is now capable of managing 6 services.

Notice that interface report_writable defines methods new_line and write, methods to be implemented by classes that can provide implementations whereby a call to one of these methods can replace the respective ABAP statement. Notice also that while method new_line has no signature, method write defines signature parameters for both a format and a value to be specified.

The unit tests of class tester still fail for the same reasons they failed with the previous version.

21.4 Exercise 146

Program: ZAUT504D

Requirements

Reason for change

- Continue to make changes to prevent Internal Session for Isolated Test Class Execution report from appearing during test.

Changes to be applied

1. In method clear_all_service_locators of class service_locator_test_helper, do the following:
   - Add the following line to the clear statement:

```
clear: o
       o
       o
     , service_locator=>singleton->report_writer
```

   - Add the following assertion at the end of the method:

```
cl_abap_unit_assert=>assert_not_bound(
  act                        = service_locator=>singleton->report_writer
  ).
```

2. Add the following test method definition to class service_factory_autester after the one for create_message_dispatcher:

```
methods      : o
               o
               o
             , create_report_writer
                 for testing
```

3. Add the following assertion to the end of method create_all_services of class service_factory_autester:

```
cl_abap_unit_assert=>assert_bound(
  act                       = service_locator=>singleton->report_writer
  ).
```

4. Add the following assertion to the end of method create_flights_organizer of class service_factory_autester:

```
cl_abap_unit_assert=>assert_not_bound(
  act                       = service_locator=>singleton->report_writer
  ).
```

5. Add the following assertion to the end of method create_flights_report of class service_factory_autester:

```
cl_abap_unit_assert=>assert_not_bound(
  act                       = service_locator=>singleton->report_writer
  ).
```

6. Add the following assertion to the end of method create_revenue_calculator of class service_factory_autester:

```
cl_abap_unit_assert=>assert_not_bound(
  act                       = service_locator=>singleton->report_writer
  ).
```

7. Add the following assertion to the end of method create_discount_calculator of class service_factory_autester:

```
cl_abap_unit_assert=>assert_not_bound(
  act                       = service_locator=>singleton->report_writer
  ).
```

8. Add the following assertion to the end of method create_message_dispatcher of class service_factory_autester:

```
cl_abap_unit_assert=>assert_not_bound(
  act                       = service_locator=>singleton->report_writer
  ).
```

9. Add the following method implementation at the end of the class:

```
method create_report_writer.
  cl_abap_unit_assert=>assert_not_bound(
    act                       = service_locator=>singleton->flights_organizer
    ).
  cl_abap_unit_assert=>assert_not_bound(
    act                       = service_locator=>singleton->flights_report
    ).
  cl_abap_unit_assert=>assert_not_bound(
    act                       = service_locator=>singleton->revenue_calculator
    ).
  cl_abap_unit_assert=>assert_initial(
    act                       = service_locator=>singleton->discount_calculator
    ).
  cl_abap_unit_assert=>assert_not_bound(
    act                       = service_locator=>singleton->message_dispatcher
    ).
  cl_abap_unit_assert=>assert_bound(
    act                       = service_locator=>singleton->report_writer
    ).
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Content of write statement appears on the screen but is preceded by red and green highlighted rows indicating the following:

```
+-------------------------------------------------------------------------------
|Internal Session for Isolated Test Class Execution
+-------------------------------------------------------------------------------
|Warning
|=======
|Program: <program name>
|Class:   <test class name>
|This window is displayed because your test case has triggered
|a list command like follows:
|- new page
|- leave to list processing
|- uline
|- write
|- new page
|- ...
|This as any interactive technique is not permitted !!
+-------------------------------------------------------------------------------
|Please avoid the use of these statements. To locate them in the source code
|setting break-points on the mentioned statements should help.
+-------------------------------------------------------------------------------
|       <content of write statement appears here>
```

Pressing Back, Exit, Cancel or ESCape results in ABAP Unit: Results Display report indicating test class tester triggers errors during methods present_report, set_alv_field_catalog, set_alv_function_module name and show_flights, and test class service_factory_autester triggers errors during methods create_all_services and create_report_writer; Status message indicates Processed: 1 program, 2 test classes, 21 test methods.

Remarks

With this version we've updated class service_locator_test_helper to accommodate the new public attribute for managing a report writer by the service locator. The methods of class service_factory_autester also have been updated to accommodate the applicable assertion against the new public attribute for managing a report writer by the service locator, including the definition of a new method create_report_writer to assert that this new attribute is bound while none of the other service locator attributes are bound.

Notice that although these test classes have been updated, neither class service_locator nor class service_factory has been changed to provide new public methods for creating and registering a report writer service.

The unit tests of class tester still fail for all the same reasons they failed with the previous version, and now unit test methods of class service_factory_autester are triggering additional unit test failures based on the changes applied to this version.

21.5 Exercise 147

Program: ZAUT504E

Requirements

Reason for change

- Continue to make changes to prevent Internal Session for Isolated Test Class Execution report from appearing during test.

Changes to be applied

1. Add the following method definition to interface service_locatable after the one for register_message_dispatcher:

```
    methods      : o
                   o
                   o
                 , register_report_writer
                     importing
                       report_writer
                         type ref
                           to report_writable
```

2. Add the following method definition to interface service_creatable after the one for create_message_dispatcher:

```
    methods      : o
                   o
                   o
                 , create_report_writer
```

3. Add the following to the aliases statement of class service_locator after the one for register_message_dispatcher:

```
    aliases      : o
                   o
                   o
                 , register_report_writer
                     for service_locatable~register_report_writer
```

4. Add the following method implementation to class service_locator after the one for register_message_dispatcher:

```
  method register_report_writer.
    me->report_writer             = report_writer.
  endmethod.
```

5. Add the following new class definition after class service_locator:

```
class report_writer                   definition
                                      final
                                      .
  public section.
    interfaces   : report_writable
                   .
    aliases      : new_line
                       for report_writable~new_line
                 , write
                     for report_writable~write
                   .
endclass.
class report_writer                   implementation.
  method new_line.
```

```
    new-line.
  endmethod.
  method write.
    constants    : default_format type sy-msgv1   value 'SY-MSGV1'
                 .
    data         : value_formatting_field
                                  type ref
                                    to data
                 .
    field-symbols: <value_formatting_field>
                                  type any
                 .
    try.
      create data value_formatting_field type (format).
    catch cx_sy_create_data_error.
      create data value_formatting_field type (default_format).
    endtry.
    if value_formatting_field     is bound.
      assign  value_formatting_field->*
          to <value_formatting_field>.
    endif.
    if <value_formatting_field>   is assigned.
      <value_formatting_field>    = value.
      write <value_formatting_field>.
    else.
      write value.
    endif.
  endmethod.
endclass.
```

6. Add the following to the aliases statement of class service_factory after the one for create_message_dispatcher:

```
    aliases      : o
                   o
                   o
                 , create_report_writer
                     for service_creatable~create_report_writer
```

7. Add the following statement to the end of method create_all_services of class service_factory:

```
    me->create_report_writer( ).
```

8. Add the following method implementation to class service_factory after the one for create_message_dispatcher:

```
  method create_report_writer.
    data         : report_writer  type ref
                                    to report_writable
                 .
    create object report_writer
             type report_writer.
    service_locator=>singleton->register_report_writer(
      exporting
        report_writer             = report_writer
        ).
  endmethod.
```

9. Add the following statement to the start of method create_report_writer of class service_factory_autester:

```
    service_factory=>singleton->create_report_writer( ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Content of write statement appears on the screen but is preceded by red and green highlighted rows indicating the following:

```
+--------------------------------------------------------------------------------
|Internal Session for Isolated Test Class Execution
+--------------------------------------------------------------------------------
|Warning
|=======
|Program: <program name>
|Class:   <test class name>
|This window is displayed because your test case has triggered
|a list command like follows:
|- new page
|- leave to list processing
|- uline
|- write
|- new page
|- ...
|This as any interactive technique is not permitted !!
+--------------------------------------------------------------------------------
|Please avoid the use of these statements. To locate them in the source code
|setting break-points on the mentioned statements should help.
+--------------------------------------------------------------------------------
|       <content of write statement appears here>
```

Pressing Back, Exit, Cancel or ESCape results in ABAP Unit: Results Display report indicating test class tester triggers errors during methods present_report, set_alv_field_catalog, set_alv_function_module_name and show_flights; Status message indicates Processed: 1 program, 2 test classes, 21 test methods.

Remarks

With this version new class report_writer has been defined and implements the report_writable interface. Also, classes service_locator and service_factory have been changed to provide new public methods for creating and registering a report writer service. In addition, unit test create_report_writer of test class service_factory_autester now creates an instance of a report writer service before asserting that the various services are correctly bound and not bound as applicable.

Notice the implementation specified for method write of class report_writer: it facilitates defining a data field using the same data type provided through the format parameter. If it is successful in defining such a field, the value provided through the value parameter is moved to that data field and that field is then used with the ABAP WRITE statement.

The unit tests of class tester still fail for all the same reasons they failed with the previous version, but none of the unit tests of class service_factory_autester are failing.

21.6 Exercise 148

Program: ZAUT504F

Requirements

Reason for change

- Continue to make changes to prevent Internal Session for Isolated Test Class Execution report from appearing during test

Changes to be applied

1. In method show_flights of class flights_report_old_format, replace the new-line and write statements with the following statements:

```
service_locator=>singleton->report_writer->new_line( ).
service_locator=>singleton->report_writer->write(
  exporting
    format                 = 'flights_organizable=>flights_row-carrid'
    value                  = conv #( flights_entry-carrid )
  ).
service_locator=>singleton->report_writer->write(
  exporting
    format                 = 'flights_organizable=>flights_row-connid'
    value                  = conv #( flights_entry-connid )
  ).
service_locator=>singleton->report_writer->write(
  exporting
    format                 = 'flights_organizable=>flights_row-fldate'
    value                  = conv #( flights_entry-fldate )
  ).
service_locator=>singleton->report_writer->write(
  exporting
    format                 = 'flights_organizable=>flights_row-price'
    value                  = conv #( flights_entry-price )
  ).
service_locator=>singleton->report_writer->write(
  exporting
    format                 = 'flights_organizable=>flights_row-currency'
    value                  = conv #( flights_entry-currency )
  ).
service_locator=>singleton->report_writer->write(
  exporting
    format                 = 'flights_organizable=>flights_row-planetype'
    value                  = conv #( flights_entry-planetype )
  ).
service_locator=>singleton->report_writer->write(
  exporting
    format                 = 'flights_organizable=>flights_row-seatsmax'
    value                  = conv #( flights_entry-seatsmax )
  ).
service_locator=>singleton->report_writer->write(
  exporting
    format                 = 'flights_organizable=>flights_row-seatsocc'
    value                  = conv #( flights_entry-seatsocc )
  ).
service_locator=>singleton->report_writer->write(
  exporting
    format                 = 'flights_organizable=>flights_row-paymentsum'
    value                  = conv #( flights_entry-paymentsum )
  ).
service_locator=>singleton->report_writer->write(
  exporting
    format                 = 'flights_organizable=>flights_row-seatsmax_b'
    value                  = conv #( flights_entry-seatsmax_b )
  ).
service_locator=>singleton->report_writer->write(
  exporting
    format                 = 'flights_organizable=>flights_row-seatsocc_b'
    value                  = conv #( flights_entry-seatsocc_b )
  ).
service_locator=>singleton->report_writer->write(
  exporting
    format                 = 'flights_organizable=>flights_row-seatsmax_f'
    value                  = conv #( flights_entry-seatsmax_f )
  ).
service_locator=>singleton->report_writer->write(
  exporting
```

```
          format                = 'flights_organizable=>flights_row-seatsocc_f'
          value                 = conv #( flights_entry-seatsocc_f )
        ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Content of write statement appears on the screen but is preceded by red and green highlighted rows indicating the following:

```
+------------------------------------------------------------------------------
|Internal Session for Isolated Test Class Execution
+------------------------------------------------------------------------------
|Warning
|=======
|Program: <program name>
|Class:   <test class name>
|This window is displayed because your test case has triggered
|a list command like follows:
|- new page
|- leave to list processing
|- uline
|- write
|- new page
|- ...
|This as any interactive technique is not permitted !!
+------------------------------------------------------------------------------
|Please avoid the use of these statements. To locate them in the source code
|setting break-points on the mentioned statements should help.
+------------------------------------------------------------------------------
|       <content of write statement appears here>
```

Pressing Back, Exit, Cancel or ESCape results in ABAP Unit: Results Display report indicating test class tester triggers errors during methods present_report, set_alv_field_catalog, set_alv_function_module_name and show_flights; Status message indicates Processed: 1 program, 2 test classes, 21 test methods.

Remarks

With this version the classical list ABAP statements NEW-LINE and WRITE have been replaced by calls their respective methods of the report writer service.

Notice that since the production code also is affected by this change, the report produced by the production code is identical to the report class flights_report_old_format was producing when it used explicit NEW-LINE and WRITE statements.

The unit tests of class tester still fail for all the same reasons they failed with the previous version,

Note: In order to avoid making this method even more complicated than it already is, the value parameter specified on the calls to method write of class report_writer use a syntax introduced relatively recently in

the life cycle of ABAP language. Here, the “CONV #([field name])” provided as the value for the “value” parameter indicates that the value in “field name” is to be converted from whatever type it is defined into the type that the can be accepted by the “value” parameter of the write method. If this technique were not used, then every field value that would be sent on a call to method write of class report_writer first would need to be moved to a field defined the same way as this “value” parameter is defined.

21.7 Exercise 149

Program: ZAUT504G

Requirements

Reason for change

- Continue to make changes to prevent Internal Session for Isolated Test Class Execution report from appearing during test.

Changes to be applied

1. Add new class report_writer_test_double after class report_writer using the following code:

```
class report_writer_test_double        definition
                                       final
                                       .
  public section.
    interfaces   : report_writable
                 .
    aliases      : new_line
                     for report_writable~new_line
                 , write
                     for report_writable~write
                 .
    data         : number_of_lines_written
                                  type int4
                 .
  private section.
    data         : new_line_pending
                                  type abap_bool value abap_true
                 .
endclass.
class report_writer_test_double        implementation.
  method new_line.
    new_line_pending              = abap_true.
  endmethod.
  method write.
    if new_line_pending           = abap_true.
      add 01 to number_of_lines_written.
      new_line_pending            = abap_false.
    endif.
  endmethod.
endclass.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Content of write statement appears on the screen but is preceded by red and green highlighted rows indicating the following:

```
+--------------------------------------------------------------------------------
|Internal Session for Isolated Test Class Execution
+--------------------------------------------------------------------------------
|Warning
|=======
|Program: <program name>
|Class:   <test class name>
|This window is displayed because your test case has triggered
|a list command like follows:
|- new page
|- leave to list processing
|- uline
|- write
|- new page
|- ...
|This as any interactive technique is not permitted !!
+--------------------------------------------------------------------------------
|Please avoid the use of these statements. To locate them in the source code
|setting break-points on the mentioned statements should help.
+--------------------------------------------------------------------------------
|       <content of write statement appears here>
```

Pressing Back, Exit, Cancel or ESCape results in ABAP Unit: Results Display report indicating test class tester triggers errors during methods present_report, set_alv_field_catalog, set_alv_function_module_name, show_flights; Status message indicates Processed: 1 program, 2 test classes, 21 test methods.

Remarks

With this version we have introduced a test double class for class report_writer: report_writer_test_double. Like class report_writer, class report_writer_test_double also implements the report_writable interface, making it interchangeable with class report_writer as the report writer service to be registered with the service locator.

Notice that the implementation of its methods do not contain any classical list statements as are found in the counterpart method implementations of class report_writer. Indeed, this class is defined as yet another test spy – a test double that is capable of recording information about its use for later reference (see xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 137).

The unit tests of class tester still fail for all the same reasons they failed with the previous version,

21.8 Exercise 150

Program: ZAUT504H

Requirements

Reason for change

- Continue to make changes to prevent Internal Session for Isolated Test Class Execution report from appearing during test.

Changes to be applied

1. In method setup of class tester:
   - Define data field messenger_test_double as reference to report_writable after the one for messenger_test_double:

```
data         : o
               o
               o
             , report_writer_test_double
                             type ref
                               to report_writable
```

   - Add the following code to the end of the method:

```
" Instantiate report_writer_test_double:
create object report_writer_test_double
         type report_writer_test_double.
" Register report_writer_test_double as report_writer service:
service_locator=>singleton->register_report_writer(
  exporting
    report_writer             = report_writer_test_double
  ).
```

2. Replace method present_report of class tester with the following code:

```
constants    : no_discount    type discount  value 00
             , alv_classic_list
                              type abap_bool value abap_false
             .
data         : report_writable
                              type ref
                                to report_writer_test_double
             .
set_bogus_message( ).
assert_message_is_bogus( ).
perform present_report using no_discount
                             alv_classic_list
                             carrier.
assert_message_not_bogus( ).
try.
  report_writable             ?= service_locator=>singleton->report_writer.
catch cx_sy_move_cast_error.
  cl_abap_unit_assert=>fail(
    msg                       = 'Caught exception in test method present_report'
    ).
endtry.
cl_abap_unit_assert=>assert_equals(
  act                         = report_writable->number_of_lines_written
  exp                         = service_locator=>singleton->flights_organizer->get_flights_count( )
  msg                         = 'Unexpected number of records written'
  ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers errors during methods set_alv_field_catalog, set_alv_function_module_name and show_flights; Status message indicates Processed: 1 program, 2 test classes, 21 test methods.

Remarks

With this version we have changed method setup of class tester to create an instance of class report_writer_test_double and to register it as the report writer service with the service locator. Method present_report of class tester also was changed to expect an instance of class reoprt_writer_test_double as the report_writer service and to use this instance to check that the number of records requested to be written matches the number of times method write of class report_writer_test_double was called.

Three of the four unit tests of class tester still fail for all the same reasons they failed with the previous version – only unit test method present_report no longer fails. Notice also that when running the unit tests the "Internal Session for Isolated Test Class Execution" list no longer appears, meaning there no longer is any manual intervention required to allow the unit tests to run to completion.

At this point we now have a program where subroutine present_report encounters the NEW-LINE and WRITE statements during its production code run but not during its unit test run. Even though some unit tests are still failing, we have effectively gained control over the classical list statements during the unit test run.

Recall that test double class report_writer_test_double, introduced with the previous version, represents test spy. Here we are accessing this test spy to perform the "Behavior Verification" (xUnit Test Patterns; G. Meszaros; 2007, Addison-Wesley; p. 468) confirming that calls to its methods have recorded the number of report lines written that we expect it should have recorded.

Let's register in our issues list that this version resolves issue #24.

#	Identified	Resolved	Description
1	ZAUT101A	ZAUT503F	No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights

#	Identified	Resolved	Description
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F	ZAUT503L	Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G	(ZAUT503L)	Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H	(ZAUT503L)	Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	**ZAUT105N**	**ZAUT504H**	**Unit test presents full screen warning upon encountering any list command (write, uline, etc.)**
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A	ZAUT402C	First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A	ZAUT402C	Second singleton class is introduced: flights_report

#	Identified	Resolved	Description
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A	ZAUT402C	Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B	ZAUT402C	Fourth singleton class is introduced: flights_report_test_double
35	ZAUT302D	ZAUT302N	Unit test method show_flights of class tester explicitly calls test double
36	ZAUT302J	ZAUT302M	Use of singleton classes result in interacting tests
37	ZAUT401A		Fifth singleton class is introduced: service_locator
38	ZAUT401A	ZAUT401G	Class service_locator defines externally changeable public attribute flights_organizer
39	ZAUT401D	ZAUT401G	Class service_locator defines externally changeable public attribute flights_report
40	ZAUT402A		Sixth singleton class is introduced: service_factory
41	ZAUT502A	ZAUT502D	Method apply_flight_discount of class tester fails to cause subroutine apply_flight_discount to call the function module

21.9 Exercise 151

Program: ZAUT504I

Requirements

Reason for change

- Fix two of the unit test failures encountered in the previous version.

Changes to be applied

1. In methods set_alv_field_catalog and set_alv_function_module_name of class tester, apply the following changes:
   - Include the following data definition:

```
data        : o
              o
              o
            , flights_report_test_double
                          type ref
```

```
                                to flights_reportable
```

- Include the following statements prior to the try statement:

```
create object flights_report_test_double
         type flights_report_test_double.
service_locator=>singleton->register_flights_report(
  exporting
    flights_report            = flights_report_test_double
  ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers error during method show_flights; Status message indicates Processed: 1 program, 2 test classes, 21 test methods.

Remarks

With this version we have resolved the unit test failures of methods set_alv_field_catalog and set_alv_function_module_name of class tester. In both cases the corresponding method creates and registers an instance of flights_report_test_double as the flights report service managed by class service_locator. This causes the specializing cast controlled by the try-endtry block to succeed and allows the corresponding asserts to be executed.

Only class tester method show_flights continues to fail.

21.10 Exercise 152

Program: ZAUT504J

Requirements

Reason for change

- Fix the remaining unit test failure encountered in the previous version.

Changes to be applied

1. Replace method show_flights of class tester with the following code:

```
data        : carrier_id_stack
                      type table
                        of s_carr_id
            , carrier_id_entry
                      like line
                        of carrier_id_stack
            , report_writable
                      type ref
                        to report_writer_test_double
            , expected_count_lines_written
```

```
                              type int4
             .
append: lufthansa             to carrier_id_stack
      , united_airlines       to carrier_id_stack
      , american_airlines     to carrier_id_stack
      .
loop at carrier_id_stack
   into carrier_id_entry.
  carrier                     = carrier_id_entry.
  call method service_locator=>singleton->flights_organizer->get_flights_via_carrier
    exporting
      carrier                 = carrier
      .
  call method service_locator=>singleton->flights_report->show_flights
    exporting
      alv_style_grid          = abap_false
    changing
      flights_stack           = service_locator=>singleton->flights_organizer->flights_stack
      .
  expected_count_lines_written
                              = expected_count_lines_written
                              + lines( service_locator=>singleton->flights_organizer->flights_stack ).
endloop.
try.
  report_writable             ?= service_locator=>singleton->report_writer.
catch cx_sy_move_cast_error.
  cl_abap_unit_assert=>fail(
    msg                       = 'Caught exception in test method show_flights'
    ).
endtry.
cl_abap_unit_assert=>assert_equals(
  act                         = report_writable->number_of_lines_written
  exp                         = expected_count_lines_written
  msg                         = 'Unexpected number of records written'
  ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 2 test classes, 21 test methods.

Remarks

With this version we have resolved the unit test failure of method show_flights of class tester by asserting against the test spy report_writer_test_double that it was called once for every row that we expect should be appearing on a report.

Once again, all unit tests pass.

22 ABAP Unit Testing 601 – Detecting Missing Service Locators

This section describes the requirements for the exercise programs associated with the Chapter 11 section titled Following the TDD Cycle in the book Automated Unit Testing with ABAP.

22.1 Exercise 153

Program: ZAUT601A

Requirements

Reason for change

- Force all services to be regarded as missing.

Changes to be applied

1. In method create_all_services of class service factory, comment out all the statements.

Run

Action: Specify Airline 'AA', no discount, ALV classic list and press Execute.

Result: Program fails with exception CX_SY_REF_IS_INITIAL.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers errors during methods present_report and show_flights and class service_factory_autester triggers error during method create_all_services; Status message indicates Processed: 1 program, 2 test classes, 21 test methods.

Remarks

With this version we have intentionally corrupted the program so that the service factory no longer registers any of the service providers with the service locator. The program fails when run in production mode.

We've done this for two reasons:

1. When no service has been registered with the service locator for one of the services it manages, we want to provide a controlled failure response, one that will alert the user where the problem was detected, rather than simply rely on the short dump screen to appear and let users fend for themselves in resolving the issue.
2. We want to become more familiar with Test-Driven Development, which is how we will approach implementing the solution for providing a controlled failure for a missing service.

Let's register in our issues list that this version introduces new issue #42.

#	Identified	Resolved	Description
1	ZAUT101A	ZAUT503F	No test for code in classic event block at-selection screen

#	Identified	Resolved	Description
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F	ZAUT503L	Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G	(ZAUT503L)	Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H	(ZAUT503L)	Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester

#	Identified	Resolved	Description
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N	ZAUT504H	Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A	ZAUT402C	First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A	ZAUT402C	Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A	ZAUT402C	Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B	ZAUT402C	Fourth singleton class is introduced: flights_report_test_double
35	ZAUT302D	ZAUT302N	Unit test method show_flights of class tester explicitly calls test double
36	ZAUT302J	ZAUT302M	Use of singleton classes result in interacting tests
37	ZAUT401A		Fifth singleton class is introduced: service_locator
38	ZAUT401A	ZAUT401G	Class service_locator defines externally changeable public attribute flights_organizer
39	ZAUT401D	ZAUT401G	Class service_locator defines externally changeable public attribute flights_report
40	ZAUT402A		Sixth singleton class is introduced: service_factory
41	ZAUT502A	ZAUT502D	Method apply_flight_discount of class tester fails to cause subroutine apply_flight_discount to call the function module

#	Identified	Resolved	Description
42	ZAUT601A		Missing services cause program exception

22.2 Exercise 154

Program: ZAUT601B

Requirements

Reason for change

- Begin applying changes to provide unit test to detect missing services.

Changes to be applied

1. Place the following interface ahead of interface report_writable:

```
interface missing_service_diagnosable.
    types        : missing_service_exception
                                type ref
                                  to cx_root
                 .
    constants    : access_to_null_service_locator
                                type string    value `Access to missing service locator occurred at`
                 .
    methods      : diagnose_missing_service
                 .
endinterface.
```

2. Place the following test class at the end of the program:

```
class missing_service_diagn_autester   definition
                                       final
                                       for testing
                                       risk level harmless
                                       duration short
                                       .
  private section.
    data         : messenger_test_double
                                type ref
                                  to messenger_test_double
                 .
    methods      : setup
                 , confirm_null_service_diagnosed
                 , diagnose_flights_organizer
                     for testing
                 , diagnose_flights_report
                     for testing
                 , diagnose_revenue_calculator
                     for testing
                 , diagnose_discount_calculator
                     for testing
                 , diagnose_report_writer
                     for testing
                 .
endclass.
class missing_service_diagn_autester   implementation.
  method setup.
    data         : service_locator_test_helper
                                type ref
                                  to service_locator_test_helper
                 .
    create object service_locator_test_helper.
    service_locator_test_helper->clear_all_service_locators( ).
```

```
    create object me->messenger_test_double.
    service_locator=>singleton->register_message_dispatcher(
      exporting
        message_dispatcher        = me->messenger_test_double
        ).
  endmethod.
  method confirm_null_service_diagnosed.
    data         : unidentified_message_entry
                                  like line
                                    of me->messenger_test_double->unidentified_message_stack
                 , contains_missing_service_text
                                  type abap_bool
                 .
    loop at me->messenger_test_double->unidentified_message_stack
       into                            unidentified_message_entry
      where type                 eq message_dispatchable=>abort_message.
      if unidentified_message_entry-text
                                  cs missing_service_diagnosable=>access_to_null_service_locator.
        contains_missing_service_text
                                  = abap_true.
      endif.
    endloop.
    cl_abap_unit_assert=>assert_equals(
      exp                         = abap_true
      act                         = contains_missing_service_text
      ).
  endmethod.
  method diagnose_flights_organizer.
    confirm_null_service_diagnosed( ).
  endmethod.
  method diagnose_flights_report.
    confirm_null_service_diagnosed( ).
  endmethod.
  method diagnose_revenue_calculator.
    confirm_null_service_diagnosed( ).
  endmethod.
  method diagnose_discount_calculator.
    confirm_null_service_diagnosed( ).
  endmethod.
  method diagnose_report_writer.
    confirm_null_service_diagnosed( ).
  endmethod.
endclass.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Program fails with exception CX_SY_REF_IS_INITIAL.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers errors during methods present_report and show_flights, class service_factory_autester triggers error during method create_all_services and class missing_service_diagn_autester triggers errors during methods diagnose_discount_calculator, diagnose_flights_organizer, diagnose_flights_report, diagnose_report_writer and diagnose_revenue_calculator; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

Test-Driven Development is based on the premise that an automated unit test is to be written first, before any corresponding production code has been written, and as a consequence the unit test should fail due to the necessary production code remaining unwritten.

With this version we have introduced a new interface named missing_service_diagnosable and a

new unit test class named missing_service_diagn_autester. The intent of the new class is to provide automated unit tests for the methods of class missing_service_diagnoser (see note below), a production code class that has yet to be written. It has unit test methods marked as “for testing” already defined for testing each of the services that class service locator manages, with the exception of the messenger service, but notice that the setup method creates and registers a messenger service using the messenger_test_double.

Notice also that we have provided method confirm_null_service_diagnosed of class missing_service_diagn_autester with an implementation to loop through the records recorded by the messenger_test_double (a test spy) to determine whether it contains a message indicating an access to a missing service. The message value being asserted in this unit test is defined as a constant in the new interface we introduced, where it is available for use by both the yet-to-be defined production class that will issue the message and the unit test method checking for the presence of it.

Each of these new unit tests currently calls method confirm_null_service_diagnosed. These now constitute legitimate failing tests since the call to confirm_null_service_diagnosed should, and does, detect that no failure message had been issued, as would be expected when production code eventually is introduced to cause the test to pass.

Many of the unit tests introduced in previous versions also fail due to the missing services the service factory is no longer registering with the service locator.

Note: Here we see an example of the naming convention I have adopted between a production class and the corresponding unit test class intended to test it. The name of the production class is “missing_service_diagnoser”. The corresponding test class normally would be “missing_service_diagnoser_autester”, but that name has 34 characters, too long for a class name in ABAP. Accordingly, I have removed from this name as many characters as necessary preceding the “_autester” suffix to arrive at a unit test class name that does not exceed the 30-character class name limit: missing_service_diagn_autester.

22.3 Exercise 155

Program: ZAUT601C

Requirements

Reason for change

- Continue applying changes to provide unit test to detect missing services.

Changes to be applied

1. Place the following singleton class after class service_locator:

```
class missing_service_diagnoser        definition
                                       final
                                       create private
                                       .
  public section.
    interfaces   : missing_service_diagnosable
                 .
    aliases      : diagnose_missing_service
                     for missing_service_diagnosable~diagnose_missing_service
                 .
    class-data   : singleton      type ref
                                    to missing_service_diagnoser
                                    read-only
                 .
    data         : missing_service_exception
                                  type missing_service_diagnosable=>missing_service_exception
```

```
                .
    class-methods: class_constructor
                .
endclass.
class missing_service_diagnoser        implementation.
  method class_constructor.
    create object missing_service_diagnoser=>singleton.
  endmethod.
  method diagnose_missing_service.
    data         : program_name   type syrepid
                 , include_name   type syrepid
                 , source_line    type i
                 , displayable_source_line
                                  type char10
                 , diagnostic     type string
                 .
    missing_service_exception->get_source_position(
      importing
        program_name             = program_name
        include_name             = include_name
        source_line              = source_line
      ).
    displayable_source_line      = source_line.
    shift displayable_source_line left deleting leading space.
    concatenate missing_service_diagnosable=>access_to_null_service_locator
                program_name
                include_name
                displayable_source_line
           into diagnostic
                  separated by space.
    if service_locator=>singleton->message_dispatcher is bound.
      service_locator=>singleton->message_dispatcher->issue_unidentified_message(
        exporting
          message_severity       = message_dispatchable=>abort_message
          text                   = diagnostic
          ).
    else.
      message diagnostic type message_dispatchable=>abort_message.
    endif.
  endmethod.
endclass.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Program fails with exception CX_SY_REF_IS_INITIAL.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers errors during methods present_report and show_flights, class service_factory_autester triggers error during method create_all_services and class missing_service_diagn_autester triggers errors during methods diagnose_discount_calculator, diagnose_flights_organizer, diagnose_flights_report, diagnose_report_writer and diagnose_revenue_calculator; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With the previous version we wrote a set of failing unit tests for which there is not yet any production code to call. With this version we have written the corresponding production code – new singleton class missing_service_diagnoser – to handle the task of providing a controlled failure response for a missing service.

There are two things to notice about the implementation of its method diagnose_missing_service:

1. It uses the instance of the exception class to retrieve the corresponding program name, include name and source code line to be used in the message it produces.
2. It checks whether the service locator has a bound instance associated with the messenger service; if so, it uses the messenger service to issue the message; if not, it uses its fallback technique of issuing the message via the ABAP MESSAGE statement.

Notice also that this new class declares public attribute missing_service_exception, defined, via type missing_service_exception of interface missing_service_diagnosable, as a reference to exception class cx_root. The intent of making this attribute public is so that it can be specified on an “into” clause on a catch statement within a try-endtry block of any external entity, which will result in this attribute holding the reference to the instance of the exception class instantiated during the exception. The corresponding “catch” clause would contain a call to method diagnose_missing_service of singleton class missing_service_diagnoser, where public attribute missing_service_exception already will be holding the reference to the instance of the corresponding exception class.

Let’s register in our issues list that this version introduces new issue #43.

#	Identified	Resolved	Description
1	ZAUT101A	ZAUT503F	No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle

#	Identified	Resolved	Description
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F	ZAUT503L	Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G	(ZAUT503L)	Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H	(ZAUT503L)	Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N	ZAUT504H	Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A	ZAUT402C	First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A	ZAUT402C	Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A	ZAUT402C	Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B	ZAUT402C	Fourth singleton class is introduced: flights_report_test_double

#	Identified	Resolved	Description
35	ZAUT302D	ZAUT302N	Unit test method show_flights of class tester explicitly calls test double
36	ZAUT302J	ZAUT302M	Use of singleton classes result in interacting tests
37	ZAUT401A		Fifth singleton class is introduced: service_locator
38	ZAUT401A	ZAUT401G	Class service_locator defines externally changeable public attribute flights_organizer
39	ZAUT401D	ZAUT401G	Class service_locator defines externally changeable public attribute flights_report
40	ZAUT402A		Sixth singleton class is introduced: service_factory
41	ZAUT502A	ZAUT502D	Method apply_flight_discount of class tester fails to cause subroutine apply_flight_discount to call the function module
42	ZAUT601A		Missing services cause program exception
43	**ZAUT601C**		**Seventh singleton class is introduced: missing_service_diagnoser**

22.4 Exercise 156

Program: ZAUT601D

Requirements

Reason for change

- Continue applying changes to provide unit test to detect missing services.

Changes to be applied

1. Replace the statement in method diagnose_flights_organizer of class missing_service_diagn_autester with the following statements:

```
data         : flights_count  type flights_organizable=>counter
             .
" given a singleton service locator
"   with all its services cleared by the setup method except for the
"   message dispatcher, which has been set to use class messenger_test_double,
try.
  " when we attempt to call a service locator that has not been established
  flights_count               = service_locator=>singleton->flights_organizer->get_flights_count( ).
catch cx_sy_ref_is_initial into missing_service_diagnoser=>singleton->missing_service_exception.
  missing_service_diagnoser=>singleton->diagnose_missing_service( ).
endtry.
" then the messenger_test_double should intercept and retain the abend message
confirm_null_service_diagnosed( ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Program fails with exception CX_SY_REF_IS_INITIAL.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers errors during methods present_report and show_flights, class service_factory_autester triggers error during method create_all_services and class missing_service_diagn_autester triggers errors during methods diagnose_discount_calculator, diagnose_flights_report, diagnose_report_writer and diagnose_revenue_calculator; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

The previous version introduced the production code capable of resolving the failing unit tests of class missing_service_diagn_autester. With this version we have provided an implementation for only one of those failing unit test methods – diagnose_flights_organizer – to test that the new production code does indeed cause the test to pass.

Notice the implementation of this unit test method. Previously it had contained only a call to method confirm_null_service_diagnosed. Now that call is preceded by the declaration of a data field defining a flights counter and a try-endtry block wrapping a call to method get_flights_count of the flights_organizer attribute of the service locator. When no flights organizer had been registered with the service locator, its flights_organizer attribute will not be bound to an instance of a class, so a call attempting to use this unbound attribute will encounter class-based exception cx_sy_ref_is_initial. This exception class is noted on the “catch” clause of the try-endtry block, and notice that its “into” clause specifies attribute missing_service_exception of the singleton class missing_service_diagnoser as the receiver of the new exception instance created during this exception processing.

Notice also that the statement following the “catch” clause is simply a call to method diagnose_missing_service of singleton class missing_service_diagnoser. As we noted with the previous version, the implementation of method diagnose_missing_service will use the message dispatch service of the service locator to issue the failure message. Method setup of class missing_service_diagn_autester creates and registers as the service locator messenger service an instance of messenger_test_double, a test spy that simply records messages but does not issue them. Accordingly, once control returns to the unit test after handling the exception, it will be method confirm_null_service_diagnosed that will be called and it asserts that the messenger test double contains the corresponding failure message.

Notice also the comments appearing in the updated implementation of this changed unit test. The recommendation for using Test-Driven Development (TDD) is to specify such comments to document the three stages of activity performed by the unit test itself:

Given	This specifies the preconditions necessary for running the unit test. In our case there are no subsequent executable statements because we have stated in the comment that the setup method already has established the necessary preconditions.
When	This specifies the action to be taken to run the unit test.
Then	This specifies the assertion(s) expected to be true after the associated action has been taken.

It is recommended to place these comments just ahead of the lines of code performing the corresponding stage of the unit test. Adhering to this discipline helps others who would be reading the test to determine what the unit test method is testing (that person may be you, years from now, well after you've had a chance to forget what you were thinking when you implemented the unit test code).

Now one of the five unit test methods defined for test class missing_service_diagn_autester passes.

22.5 Exercise 157

Program: ZAUT601E

Requirements

Reason for change

- Continue applying changes to provide unit test to detect missing services.

Changes to be applied

1. Replace the statement in method diagnose_flights_report of class missing_service_diagn_autester with the following statements:

```
data         : flights_stack  type flights_organizable=>flights_list
             .
" given a singleton service locator
"   with all its services cleared by the setup method except for the
"   message dispatcher, which has been set to use class messenger_test_double,
try.
  " when we attempt to call a service locator that has not been established
  call method service_locator=>singleton->flights_report->show_flights
    exporting
      alv_style_grid          = abap_false
    changing
      flights_stack           = flights_stack
      .
catch cx_sy_ref_is_initial into missing_service_diagnoser=>singleton->missing_service_exception.
  missing_service_diagnoser=>singleton->diagnose_missing_service( ).
endtry.
" then the messenger_test_double should intercept and retain the abend message
confirm_null_service_diagnosed( ).
```

Run

Action: Specify Airline 'AA', no discount, ALV classic list and press Execute.

Result: Program fails with exception CX_SY_REF_IS_INITIAL.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers errors during methods present_report and show_flights, class service_factory_autester triggers error during method create_all_services and class missing_service_diagn_autester triggers errors during methods diagnose_discount_calculator, diagnose_report_writer and diagnose_revenue_calculator; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With this version we have provided an implementation for another one of the failing unit test methods of class class missing_service_diagn_autester service: diagnose_flights_report.

This unit test contains an implementation similar to that of the unit test changed with the previous version – specifically, it contains a try-endtry block with a “catch” clause to intercept class-based exception cx_sy_ref_is_initial wrapping a call to a service provided by the service locator.

Now two of the five unit test methods defined for test class missing_service_diagn_autester pass.

22.6 Exercise 158

Program: ZAUT601F

Requirements

Reason for change

- Continue applying changes to provide unit test to detect missing services.

Changes to be applied

1. Replace the statement in method diagnose_revenue_calculator of class missing_service_diagn_autester with the following statements:

```
data         : flight_revenue
                          type flights_organizable=>flights_row-paymentsum
               .
" given a singleton service locator
"   with all its services cleared by the setup method except for the
"   message dispatcher, which has been set to use class messenger_test_double,
try.
  " when we attempt to call a service locator that has not been established
  call method service_locator=>singleton->revenue_calculator->get_flight_revenue
    exporting
      fare_price              = 00
      number_of_passengers    = 00
    importing
      flight_revenue          = flight_revenue
      .
catch cx_sy_ref_is_initial into missing_service_diagnoser=>singleton->missing_service_exception.
  missing_service_diagnoser=>singleton->diagnose_missing_service( ).
endtry.
" then the messenger_test_double should intercept and retain the abend message
confirm_null_service_diagnosed( ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Program fails with exception CX_SY_REF_IS_INITIAL.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers errors during methods present_report and show_flights, class service_factory_autester triggers error during method create_all_services and class missing_service_diagn_autester triggers errors during methods diagnose_discount_calculator and diagnose_report_writer; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With this version we have provided an implementation for another one of the failing unit test methods of class class missing_service_diagn_autester service: diagnose_revenue_calculator.

This unit test contains an implementation similar to that of the unit test changed with the previous version – specifically, it contains a try-endtry block with a “catch” clause to intercept class-based exception cx_sy_ref_is_initial wrapping a call to a service provided by the service locator.

Now three of the five unit test methods defined for test class missing_service_diagn_autester pass.

22.7 Exercise 159

Program: ZAUT601G

Requirements

Reason for change

- Continue applying changes to provide unit test to detect missing services.

Changes to be applied

1. Replace the statement in method diagnose_discount_calculator of class missing_service_diagn_autester with the following statements:

```
data         : discount_fare
                        type flights_organizable=>flights_row-price
               .
" given a singleton service locator
"   with all its services cleared by the setup method except for the
"   message dispatcher, which has been set to use class messenger_test_double,
try.
  " when we attempt to call a service locator that has not been established
  call function service_locator=>singleton->discount_calculator
    exporting
      full_fare               = 00
      discount                = 00
    importing
      discount_fare           = discount_fare
    exceptions
      others                  = 0
      .
catch cx_sy_dyn_call_illegal_func into missing_service_diagnoser=>singleton->missing_service_exception.
  missing_service_diagnoser=>singleton->diagnose_missing_service( ).
endtry.
" then the messenger_test_double should intercept and retain the abend message
confirm_null_service_diagnosed( ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Program fails with exception CX_SY_REF_IS_INITIAL.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers errors during methods present_report and show_flights, class service_factory_autester triggers error during method create_all_services and class missing_service_diagn_autester triggers error during method diagnose_report_writer; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With this version we have provided an implementation for another one of the failing unit test methods of class class missing_service_diagn_autester service: diagnose_discount_calculator.

This unit test contains an implementation similar to that of the unit test changed with the previous version – specifically, it contains a try-endtry block with a “catch” clause to intercept a class-based exception wrapping a call to a service provided by the service locator. The significant difference here is that the class-based exception raised is not type cx_sy_ref_is_initial but type cx_sy_dyn_call_illegal_func, because this service is provided not by an instance of a class but by the name of a function module.

Now four of the five unit test methods defined for test class missing_service_diagn_autester pass.

22.8 Exercise 160

Program: ZAUT601H

Requirements

Reason for change

- Continue applying changes to provide unit test to detect missing services.

Changes to be applied

1. Replace the statement in method diagnose_report_writer of class missing_service_diagn_autester with the following statements:

```
" given a singleton service locator
"   with all its services cleared by the setup method except for the
"   message dispatcher, which has been set to use class messenger_test_double,
try.
  " when we attempt to call a service locator that has not been established
  service_locator=>singleton->report_writer->new_line( ).
catch cx_sy_ref_is_initial into missing_service_diagnoser=>singleton->missing_service_exception.
  missing_service_diagnoser=>singleton->diagnose_missing_service( ).
endtry.
" then the messenger_test_double should intercept and retain the abend message
confirm_null_service_diagnosed( ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Program fails with exception CX_SY_REF_IS_INITIAL.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers errors during methods present_report and show_flights, and class service_factory_autester triggers error during method create_all_services; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With this version we have provided an implementation for the last failing unit test method of class class missing_service_diagn_autester service: diagnose_report_writer.

This unit test contains an implementation similar to that of the unit test changed with previous versions – specifically, it contains a try-endtry block with a “catch” clause to intercept class-based exception cx_sy_ref_is_initial wrapping a call to a service provided by the service locator.

Now all five unit test methods defined for test class missing_service_diagn_autester pass.

22.9 Exercise 161

Program: ZAUT601I

Requirements

Reason for change

- Begin applying changes to prevent program exception failures.

Changes to be applied

1. Change the at select-screen event block in the following way:
   - Place the following statement ahead of the perform statement:

```
try.
```

   - Place the following statements after the perform statement:

```
catch cx_sy_ref_is_initial into missing_service_diagnoser=>singleton->missing_service_exception.
  missing_service_diagnoser=>singleton->diagnose_missing_service( ).
endtry.
```

2. Add the following clause to the signature of subroutine process_selection:

```
                raising cx_sy_ref_is_initial.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Cancel popup message window appears showing the program name, include name and statement number where an attempt to access a missing service locator was intercepted.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers errors during methods present_report and show_flights, and class service_factory_autester triggers error during method create_all_services; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With this version we have included in the signature of subroutine process_selection a raising parameter for exception cx_sy_ref_is_initial. We've also surrounded the perform statement to this subroutine in the "at selection screen" classic event block with a try-endtry block to catch this exception, and when caught to invoke method diagnose_missing_service of class missing_service_diagnoser.

The program no longer fails when run in production mode, but instead presents a cancellation message indicating where the exception was caught.

22.10 Exercise 162

Program: ZAUT601J

Requirements

Reason for change

- Fix the problem diagnosed in the preceding version.

Changes to be applied

1. Uncomment the call to method create_flights_organizer in method create_all_services of service_factory.

Run

Action: Specify Airline 'AA', no discount, ALV classic list and press Execute.

Result: Program fails with exception CX_SY_REF_IS_INITIAL.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers errors during methods present_report and show_flights, and class service_factory_autester triggers error during method create_all_services; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With this version we have reactivated the line of code in method create_all_services of class service_factory to create a flights organizer service.

Again the program fails when run in production mode, but at a different location than the failure identified by the previous version.

22.11 Exercise 163

Program: ZAUT601K

Requirements

Reason for change

- Continue applying changes to prevent program exception failures.

Changes to be applied

1. Change the at end-of-selection event block in the following way:
   - Place the following statement ahead of the perform statement:

```
try.
```

   - Place the following statements after the perform statement:

```
catch cx_sy_ref_is_initial into missing_service_diagnoser=>singleton->missing_service_exception.
  missing_service_diagnoser=>singleton->diagnose_missing_service( ).
endtry.
```

2. Add the following clause to the signatures of subroutines present_report and adjust_flight_revenue:

```
raising cx_sy_ref_is_initial.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Cancel popup message window appears showing the program name, include name and statement number where an attempt to access a missing service locator was intercepted.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers errors during methods present_report and show_flights, and class service_factory_autester triggers error during method create_all_services; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With this version we have included in the signatures of subroutines present_report and adjust_flight_revenue a raising parameter for exception cx_sy_ref_is_initial. We’ve also surrounded the perform statement to subroutine present_report in the “end-of-selection” classic event block with a try-endtry block to catch this exception, and when caught to invoke method diagnose_missing_service of class missing_service_diagnoser.

Notice that subroutine adjust_flight_revenue is called by subroutine present_report. Accordingly, placing the raising parameter on the signature of subroutine adjust_flight_revenue enables an exception raised here to be propagated back to its caller, and placing the raising parameter on the signature of subroutine present_report enables an exception raised here to be propagated back to its caller, which in this case is the “end-of-selection” classic event block where the try-endtry block facilitates catching and processing this exception.

The program no longer fails when run in production mode, but instead presents a cancellation message indicating where the exception was caught.

22.12 Exercise 164

Program: ZAUT601L

Requirements

Reason for change

- Fix the problem diagnosed in the preceding version.

Changes to be applied

1. Uncomment the call to method create_revenue_calculator in method create_all_services of service_factory.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Program fails with exception CX_SY_REF_IS_INITIAL.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers errors during methods present_report and show_flights, and class service_factory_autester triggers error during method create_all_services; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With this version we have reactivated the line of code in method create_all_services of class service_factory to create a flight revenue calculator service.

Again the program fails when run in production mode, but at a different location than the failure identified by the previous version.

Perhaps you can see the pattern we are pursuing here: Enable the production program to diagnose a missing service, then reactivate that service in method create_all_services of class service_factory, whereupon the program then fails with the next uncaught missing service.

22.13 Exercise 165

Program: ZAUT601M

Requirements

Reason for change

- Continue applying changes to prevent program exception failures.

Changes to be applied

1. Add the following clause to the signature of subroutine show_flights_count:

```
raising cx_sy_ref_is_initial.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Cancel popup message window appears showing the program name, include name and statement number where an attempt to access a missing service locator was intercepted.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers errors during methods present_report and show_flights, and class service_factory_autester triggers error during method create_all_services; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With this version we have included in the signature of subroutine show_flights_count a raising parameter for exception cx_sy_ref_is_initial. Notice that subroutine show_flights_count is called by subroutine present_report, which already has had its signature changed to raise this exception to its caller.

The program no longer fails when run in production mode, but instead presents a cancellation message indicating where the exception was caught.

22.14 Exercise 166

Program: ZAUT601N

Requirements

Reason for change

- Fix the problem diagnosed in the preceding version.

Changes to be applied

1. Uncomment the call to method create_message_dispatcher in method create_all_services of service_factory.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Cancel popup message window appears showing the program name, include name and statement number where an attempt to access a missing service locator was intercepted.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers errors during methods present_report and show_flights, and class service_factory_autester triggers error during method create_all_services; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With this version we have reactivated the line of code in method create_all_services of class service_factory to create a message dispatch service.

Again the program presents a cancellation message indicating where the exception was caught, now at a different location than the previous version.

22.15 Exercise 167

Program: ZAUT601O

Requirements

Reason for change

- Fix the problem diagnosed in the preceding version.

Changes to be applied

1. Uncomment the call to method create_flights_report in method create_all_services of service_factory.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Program fails with exception CX_SY_REF_IS_INITIAL.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class service_factory_autester triggers error during method create_all_services; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With this version we have reactivated the line of code in method create_all_services of class service_factory to create a flights report service.

Again the program fails when run in production mode, but at a different location than the failure identified by the previous version.

22.16 Exercise 168

Program: ZAUT601P

Requirements

Reason for change

- Continue applying changes to prevent program exception failures.

Changes to be applied

1. Add the following clause to the signature of method show_flights of interface flights_reportable:

```
raising
  cx_sy_ref_is_initial
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Cancel popup message window appears showing the program name, include name and statement number where an attempt to access a missing service locator was intercepted.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class service_factory_autester triggers error during method create_all_services; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With this version we have included in the signature of method show_flights of interface flights_reportable a raising parameter for exception cx_sy_ref_is_initial.

The program no longer fails when run in production mode, but instead presents a cancellation message indicating where the exception was caught.

22.17 Exercise 169

Program: ZAUT601Q

Requirements

Reason for change

- Fix the problem diagnosed in the preceding version.

Changes to be applied

1. Uncomment the call to method create_report_writer in method create_all_services of service_factory.

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Action: Specify Airline ‘AA’, **discount 10,** ALV classic list and press Execute.

Result: Program fails with exception CX_SY_DYN_CALL_ILLEGAL_FUNC.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class service_factory_autester triggers error during method create_all_services; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With this version we have reactivated the line of code in method create_all_services of class service_factory to create a report writer service.

When the program is run in production mode with no discount value specified it presents the flights report. However, when run in production mode with a discount value specified the program fails. Notice that the exception is not the same one we have seen so far with missing services (CX_SY_REF_IS_INITIAL) when the service is provided by an instance of a class, but now it is an exception associated with a dynamic call to a function module (CX_SY_DYN_CALL_ILLEGAL_FUNC).

22.18 Exercise 170

Program: ZAUT601R

Requirements

Reason for change

- Continue applying changes to prevent program exception failures.

Changes to be applied

1. Change the catch clause in the end-of-selection event block from ...

```
catch cx_sy_ref_is_initial into missing_service_diagnoser=>singleton->missing_service_exception.
```

... to:

```
catch cx_sy_dyn_call_illegal_func
      cx_sy_ref_is_initial into missing_service_diagnoser=>singleton->missing_service_exception.
```

2. Change the raising clause of subroutine present_report from ...

```
        raising cx_sy_ref_is_initial.
```

… to:

```
        raising cx_sy_ref_is_initial
```

```
cx_sy_dyn_call_illegal_func.
```

3. Add the following clause to the signature of subroutine apply_flight_discount:

```
raising cx_sy_dyn_call_illegal_func.
```

Run

Action: Specify Airline ‘AA’, **discount 10**, ALV classic list and press Execute.

Result: Cancel popup message window appears showing the program name, include name and statement number where an attempt to access a missing service locator was intercepted.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class service_factory_autester triggers error during method create_all_services; Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With this version we have included in the signatures of subroutines present_report and apply_flight_discount a raising parameter for exception cx_sy_dyn_call_illegal_func. Also, we have included this exception in the catch clause of the try-endtry block surrounding the call to subroutine present_report in the “end-of-selection” classic event block.

The program no longer fails when run in production mode when a discount is specified, but instead presents a cancellation message indicating where the exception was caught.

22.19 Exercise 171

Program: ZAUT601S

Requirements

Reason for change

- Fix the problem diagnosed in the preceding version.

Changes to be applied

1. Uncomment the call to method create_discount_calculator in method create_all_services of service_factory.

Run

Action: Specify Airline ‘AA’, **discount 10**, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 3 test classes, 26 test methods.

Remarks

With this version we have reactivated the line of code in method create_all_services of class service_factory to create a flight discount calculation service. At this point all of the statements in method create_all_services of class service_factory that had been commented out in version ZAUT601A have been reactivated.

Now when the program is run in production mode with a discount specified it runs to completion, and all of the unit tests now pass.

Let's register in our issues list that this version resolves issue #42.

#	Identified	Resolved	Description
1	ZAUT101A	ZAUT503F	No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle

#	Identified	Resolved	Description
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT
18	ZAUT105F	ZAUT503L	Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G	(ZAUT503L)	Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H	(ZAUT503L)	Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N	ZAUT504H	Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A	ZAUT402C	First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A	ZAUT402C	Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A	ZAUT402C	Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B	ZAUT402C	Fourth singleton class is introduced: flights_report_test_double

#	Identified	Resolved	Description
35	ZAUT302D	ZAUT302N	Unit test method show_flights of class tester explicitly calls test double
36	ZAUT302J	ZAUT302M	Use of singleton classes result in interacting tests
37	ZAUT401A		Fifth singleton class is introduced: service_locator
38	ZAUT401A	ZAUT401G	Class service_locator defines externally changeable public attribute flights_organizer
39	ZAUT401D	ZAUT401G	Class service_locator defines externally changeable public attribute flights_report
40	ZAUT402A		Sixth singleton class is introduced: service_factory
41	ZAUT502A	ZAUT502D	Method apply_flight_discount of class tester fails to cause subroutine apply_flight_discount to call the function module
42	**ZAUT601A**	**ZAUT601S**	**Missing services cause program exception**
43	ZAUT601C		Seventh singleton class is introduced: missing_service_diagnoser

23 ABAP Unit Testing 701 – Using the ABAP Test Double Framework

This section describes the requirements for the exercise programs associated with the Chapter 12 section titled ABAP Test Double Framework in the book Automated Unit Testing with ABAP.

23.1 Exercise 172

Program: ZAUT701A

Requirements

Reason for change

- Begin the process of implementing a unit test using the ABAP Test Double Framework.

Changes to be applied

1. Add a new unit test method adjust_flight_revenue_via_atdf to class tester:
   - Add method definition for adjust_flight_revenue_via_atdf to the private section of class tester following the definition for method adjust_flight_revenue:

```
  methods      : o
                 o
                 o
               , adjust_flight_revenue_via_atdf
                   for testing
```

   - Include the following method implementation after the implementation for method adjust_flight_revenue:

```
method adjust_flight_revenue_via_atdf.
  constants    : revenue_value_12345
                               type flights_organizable=>flights_row-paymentsum
                                            value 12345
               .
  data         : revenue_calculator_test_double
                               type ref
                                 to zif_flight_revenue_calculable
               , test_double_configurer
                               type ref
                                 to if_abap_testdouble_config
               , flight_revenue type flights_organizable=>flights_row-paymentsum
               .
  " Given a test double instantiated using the ABAP Test Double Framework (ATDF) to recognize
  "   the methods defined for global interface zif_flight_revenue_calculable ...
  revenue_calculator_test_double
                               ?= cl_abap_testdouble=>create( 'zif_flight_revenue_calculable' ).
  " ... and configured to supply the following value for calls to methods defined with a parameter
  "   named flight_revenue ...
  test_double_configurer       = cl_abap_testdouble=>configure_call( revenue_calculator_test_double ).
  test_double_configurer       = test_double_configurer->set_parameter(
                                   name  = 'flight_revenue'
                                   value = revenue_value_12345
                                   ).
  " ... and configured to associate the following calling parameters and their respective
  "   values for a call to method get_flight_revenue with the parameter configured in the
  "   previous statement
  "   (notice with this call that the flight_revenue parameter of the get_flights_revenue
  "   method is not specified) ...
  revenue_calculator_test_double->get_flight_revenue(
    exporting
      fare_price               = 00
      number_of_passengers     = 00
    ).
  " When the ATDF test double receives a call to the method configured in the previous statement
```

```
    "   with calling parameter values matching those configured in the previous statement
    "   (notice with this call that the flight_revenue parameter of the get_flights_revenue
    "   method is specified) ...
    revenue_calculator_test_double->get_flight_revenue(
      exporting
        fare_price               = 00
        number_of_passengers     = 00
      importing
        flight_revenue           = flight_revenue
      ).
    " Then the value imported into field flight_revenue should match the value
    "   used to configure the test double with the parameter named flight_revenue:
    cl_abap_unit_assert=>assert_equals(
      act                        = flight_revenue + 01
      exp                        = revenue_value_12345
      msg                        = 'Flight revenue value other than expected'
      ).
  endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers error during method adjust_flight_revenue_via_atdf indicating message "Flight revenue value other than expected"; Status message indicates Processed: 1 program, 3 test classes, 27 test methods.

Remarks

With this version we have added new unit test method adjust_flight_revenue_via_atdf to class tester for determining how we might write a unit test for subroutine adjust_flight_revenue if we were to use the ABAP Test Double Framework instead of an instance of test double zcl_revenue_calc_test_double as is currently used with unit test method adjust_flight_revenue. Classes zcl_flight_revenue_calculator and zcl_revenue_calc_test_double are the only candidates applicable for substitution by a test double defined through the ABAP Test Double Framework because these are the only entities in our program to use a global interface.

Note: At the time of this writing, the ABAP Test Double Framework (ATDF) works only with global interfaces. Its lack of support for both local and global classes does not present much of an impediment because classes always can implement an interface, and indeed this is considered a best practice for making public methods available to classes.

It is, however, unfortunate that the ATDF does not support local interfaces, which does present an impediment for using it with local classes implementing local interfaces. Since the bulk of the classes used by the example programs are local classes implementing local interfaces, it would suggest that the ATDF is of only limited usefulness with them. It is hoped that SAP will extend to the ATDF the capability to enable its use with local interfaces, making it a much more attractive test doubling facility for use with automated unit tests.

Eventually we will have this new unit test calling subroutine adjust_flight_revenue, just as unit test method adjust_flight_revenue does, but at this point we merely are trying to determine whether we can establish a test where the ATDF is configured for invoking the only method defined for interface zif_flight_revenue_calculable – get_flight_revenue – and for it to provide a result. Accordingly, notice that we have done the following with this unit test:

1. Instantiated an instance of a class implementing interface zif_flight_revenue_calculable. This was done with the statement calling static method create of class cl_abap_testdouble, which returns an instance of if_abap_testdouble_config to our reference variable test_double_configurer.
2. Configured the test double to provide the value 12345 for the parameter flight_revenue when method get_flight_revenue is called with parameters fare_price equal 00 and number_of_passengers equal 00. This was done with the three statements following the statement calling static method create of class cl_abap_testdouble: The first two of those statements facilitate providing the answer and the third statement facilitates the name of the method and its associated parameter values that should cause that answer to be provided. Effectively, this constitutes setting the answer first and the question associated with that answer second. That is, this instance of the ATDF test double has been configured to provide the answer 12345 to parameter flight_revenue whenever its method get_flights_revenue is invoked with parameters flight_revenue = 0 and number_of_passengers = 0.
3. Called method get_flights_revenue of the ATDF test double, specifying the two parameters and their respective values configured to provide the answer 12345 into field flight_revenue.
4. Asserted that the actual answer provided by the call to the ATDF test double is the expected answer. In this case, notice that we have indicated to increase the “act” parameter by 1 on the call to method assert_equals of class cl_abap_unit_assert, which forces an assertion failure.

23.2 Exercise 173

Program: ZAUT701B

Requirements

Reason for change

- Fix the problem diagnosed in the preceding version.

Changes to be applied

1. In unit test method adjust_flight_revenue_via_atdf of class tester, change the "act" parameter on the call to method cl_abap_unit_assert=>assert_equal from ...

```
act                     = flight_revenue + 01
```

... to

```
act                     = flight_revenue
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 3 test classes, 27 test methods.

Remarks

With this version we have removed the increment of the “act” parameter on the call to method assert_equals of class cl_abap_unit_assert, enabling the unit test to pass. We now have a working example of:

1. instantiating and configuring the ATDF test double to recognize the methods defined in a global interface
2. loading it with an response to be supplied to a yet-to-be-determined request
3. registering with it the request that should trigger that response
4. invoking upon it that registered request
5. and finally asserting that it had provided the correct response

23.3 Exercise 174

Program: ZAUT701C

Requirements

Reason for change

- Continue the process of implementing a unit test using the ABAP Test Double Framework.

Changes to be applied

1. Apply the following changes to unit test method adjust_flight_revenue_via_atdf of class tester:
   - Remove the data definition for test_double_configurer.
   - Replace the following two statements ...

```
test_double_configurer        = cl_abap_testdouble=>configure_call( revenue_calculator_test_double ).
test_double_configurer        = test_double_configurer->set_parameter(
                                    name  = 'flight_revenue'
                                    value = revenue_value_12345
                                    ).
```

... with the following single chained method call statement:

```
cl_abap_testdouble=>configure_call( revenue_calculator_test_double
  )->set_parameter( name  = 'flight_revenue'
                    value = revenue_value_12345
  ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 3 test classes, 27 test methods.

Remarks

With this version we have removed the definition of field test_double_configurer and consolidated the two statements referencing it into a single chained method call. This not only removes an unnecessary data definition but, more important, allows us to configure the ATDF test double using what is known as the “builder” design

pattern, which the ATDF configuration classes have been defined to enable. We shall see an example later where a single chained method call statement to configure an ATDF test double includes a variety of configuration settings, which effectively enable the test double to be “built” on the fly using a single chained method call.

23.4 Exercise 175

Program: ZAUT701D

Requirements

Reason for change

- Continue the process of implementing a unit test using the ABAP Test Double Framework.

Changes to be applied

1. Replace the implementation of unit test method adjust_flight_revenue_via_atdf of class tester with the following statements:

```
method adjust_flight_revenue_via_atdf.
  constants    : revenue_value_12345
                               type flights_organizable=>flights_row-paymentsum
                                            value 12345
               .
  data         : revenue_calculator_test_double
                               type ref
                                 to zif_flight_revenue_calculable
               , flights_entry  like line
                                 of service_locator=>singleton->flights_organizer->flights_stack
               .
  " Given a test double instantiated using the ABAP Test Double Framework (ATDF) to recognize
  "   the methods defined for global interface zif_flight_revenue_calculable ...
  revenue_calculator_test_double
                               ?= cl_abap_testdouble=>create( 'zif_flight_revenue_calculable' ).
  " ... and configured to supply the following value for calls to methods defined with a parameter
  "   named flight_revenue ...
  cl_abap_testdouble=>configure_call( revenue_calculator_test_double
    )->set_parameter( name  = 'flight_revenue'
                      value = revenue_value_12345
    ).
  " ... and configured to associate the following calling parameters and their respective
  "   values for a call to method get_flight_revenue with the parameter configured in the
  "   previous statement
  "   (notice with this call that the flight_revenue parameter of the get_flights_revenue
  "   method is not specified) ...
  revenue_calculator_test_double->get_flight_revenue(
    exporting
      fare_price               = 00
      number_of_passengers     = 00
    ).
  " ... and this ATDF test double registered as the revenue calculator service ...
  service_locator=>singleton->register_revenue_calculator(
    exporting
      revenue_calculator       = revenue_calculator_test_double
      ).
  " When the adjust_flight_revenue subruoutine is called ...
  perform adjust_flight_revenue changing service_locator=>singleton->flights_organizer->flights_stack.
  " Then the revenue calculated for each flight should match the value
  "   used to configure the test double with the parameter named flight_revenue:
  loop at service_locator=>singleton->flights_organizer->flights_stack
     into flights_entry.
    cl_abap_unit_assert=>assert_equals(
      act                      = flights_entry-paymentsum
      exp                      = revenue_value_12345
      msg                      = 'Flight revenue value other than expected'
      ).
  endloop.
endmethod.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers error during method adjust_flight_revenue_via_atdf indicating message "Flight revenue value other than expected"; Status message indicates Processed: 1 program, 3 test classes, 27 test methods.

Remarks

With this version we have adjusted unit test method adjust_flight_revenue_via_atdf of class tester to model unit test adjust_flight_revenue – specifically, after configuring the ATDF test double as we had in the previous version, it is registered with the service locator to provide the service for calculating flight revenue, then a call is made to subroutine adjust_flight_revenue followed by a loop through all the records of service_locator=>singleton->flights_organizer->flights_stack to assert that the flight revenue value for each row is the same as the value we used to configure the ATDF test double. Notice that unit test method adjust_flight_revenue_via_atdf ends with both a perform statement followed by a loop construct, statements virtually identical to the way unit test method adjust_flight_revenue ends.

23.5 Exercise 176

Program: ZAUT701E

Requirements

Reason for change

- Begin the process of fixing the problem diagnosed in the preceding version.

Changes to be applied

1. Apply the following changes to unit test method adjust_flight_revenue_via_atdf of class tester:
   - Add data definition failure_message after the one for revenue_calculator_test_double:

```
data        : o
              o
              o
            , failure_message
                           type string
```

   - Place the following statements as the first statements in the loop construct:

```
  failure_message            = sy-tabix.
  shift failure_message left deleting leading space.
  concatenate 'Flight revenue value other than expected on flights entry'
              failure_message
         into failure_message
                separated by space.
```

- Replace the msg parameter of the call to method assert_equals of class cl_abap_unit_assert with:

```
msg                    = failure_message
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers error during method adjust_flight_revenue_via_atdf indicating message "Flight revenue value other than expected on flights entry 1"; Status message indicates Processed: 1 program, 3 test classes, 27 test methods.

Remarks

With this version we have added some clarity to the failure encountered in unit test method adjust_flight_revenue_via_atdf. We want to know which of the rows of service_locator=>singleton->flights_organizer->flights_stack causes the failure. With this added code it becomes clear that the failure occurs with the very first row retrieved by the loop statement.

23.6 Exercise 177

Program: ZAUT701F

Requirements

Reason for change

- Complete the process of fixing the problem diagnosed in the preceding version.

Changes to be applied

1. Apply the following changes to unit test method adjust_flight_revenue_via_atdf of class tester:
   - Replace the compound method call statement and its preceding 2 comment lines with the following:

```
" ... and configured to supply the following value for calls to methods defined with a parameter
"   named flight_revenue, ignoring any parameters specified ...
cl_abap_testdouble=>configure_call( revenue_calculator_test_double
  )->set_parameter( name  = 'flight_revenue'
                    value = revenue_value_12345
  )->ignore_all_parameters(
  ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 3 test classes, 27 test methods.

Remarks

With this version we have corrected the failure encountered by the previous version. Although we configured the ATDF test double to provide the correct response, we had indicated that response should be issued when method get_flights_revenue is invoked with parameters fare_price = 00 and number_of_passengers = 00. However, this method call is being made from subroutine adjust_flight_revenue and it is not using zero for the value of these two parameters.

Accordingly, we have changed the configuration of the ATDF test double to indicate that we want its response to be issued irrespective of the values specified for any parameters used to call method get_flights_revenue of the test double. This is achieved by including the call to method “ignore_all_parameters” on the chained method call to configure the ATDF test double. Notice that we have added one more consideration for “building” the ATDF test double and it was easy to include it in the chained method call.

23.7 Exercise 178

Program: ZAUT701G

Requirements

Reason for change

- Continue the process of implementing a unit test using the ABAP Test Double Framework.

Changes to be applied

1. Apply the following changes to unit test method adjust_flight_revenue_via_atdf of class tester:
   - Add constants definition revenue_value_23456 after the one for revenue_value_12345:

```
constants    : o
               o
               o
             , revenue_value_23456
                            type flights_organizable=>flights_row-paymentsum
                                          value 23456
```

   - Replace the compound method call statement and its preceding 2 comment lines with the following set of statements:

```
" ... and configured to supply the following value for the first 2 calls to methods defined with a parameter
"   named flight_revenue, ignoring any parameters specified ...
cl_abap_testdouble=>configure_call( revenue_calculator_test_double
  )->set_parameter( name  = 'flight_revenue'
                    value = revenue_value_12345
  )->times( 2
  )->ignore_all_parameters(
  ).
" ... and configured to associate the following calling parameters and their respective
"   values for a call to method get_flight_revenue with the parameter configured in the
"   previous statement
"   (notice with this call that the flight_revenue parameter of the get_flights_revenue
"   method is not specified) ...
revenue_calculator_test_double->get_flight_revenue(
  exporting
```

```
    fare_price               = 00
    number_of_passengers     = 00
  ).
" ... and configured to supply the following value for any subsequent calls to methods defined with a parameter
"   named flight_revenue, ignoring any parameters specified ...
cl_abap_testdouble=>configure_call( revenue_calculator_test_double
  )->set_parameter( name  = 'flight_revenue'
                    value = revenue_value_23456
  )->ignore_all_parameters(
  ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers error during method adjust_flight_revenue_via_atdf indicating message "Flight revenue value other than expected on flights entry 3"; Status message indicates Processed: 1 program, 3 test classes, 27 test methods.

Remarks

With this version we are exploring the various options available to us in using a test double created via the ATDF. Here we have indicated that for the first 2 calls to method get_flight_revenue we want the value returned via the flight_revenue parameter to be 12345, and then for any subsequent calls to that method we want the value returned via the flight_revenue parameter to be 23456.

Notice that this now took multiple calls to complete the configuration: one set of configuration settings provides the response 12345 to the first 2 calls to method get_flight_revenue; the second set of configuration settings provides the response 23456 to any subsequent calls to the same method. Notice also that yet again we have added to the first chained method call one more consideration for “building” the ATDF test double by including the call to method “times” and indicating that this response should be provided with the first 2 calls to to method get_flight_revenue, and again it was easy to include this in the chained method call.

23.8 Exercise 179

Program: ZAUT701H

Requirements

Reason for change

- Fix the problem diagnosed in the preceding version.

Changes to be applied

1. Apply the following changes to unit test method adjust_flight_revenue_via_atdf of class tester:
   - Replace the call to method assert_equals of class cl_abap_unit_assert with the following set of statements:

```
if sy-tabix le 02.
```

```
    " The first 2 calls to the ATDF test double for revenue calculator will cause the
    "   following value to be returned:
    cl_abap_unit_assert=>assert_equals(
      act                     = flights_entry-paymentsum
      exp                     = revenue_value_12345
      msg                     = failure_message
      ).
  else.
    " Subsequent calls to the ATDF test double for revenue calculator will cause the
    "   following value to be returned:
    cl_abap_unit_assert=>assert_equals(
      act                     = flights_entry-paymentsum
      exp                     = revenue_value_23456
      msg                     = failure_message
      ).
  endif.
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 3 test classes, 27 test methods.

Remarks

With this version version we have fixed the unit test failure encountered by the previous version. Since we had configured the ATDF test double to respond with 12345 for only the first two calls to get_flight_revenue, then we should expect that only the first two rows of service_locator=>singleton->flights_organizer->flights_stack will contain this value.

Sure enough, the failure message indicates that the failure occurs on the 3rd row of this internal table, a row we should expect to contain the value 23456 we had used to configure the ATDF test double for all calls beyond the first 2 calls to method get_flight_revenue. Accordingly, this change checks the row number for which we are asserting the value: only the first two rows should be checked for value 12345; subsequent rows should be checked for value 23456. Now the unit test passes again.

23.9 Exercise 180

Program: ZAUT701I

Requirements

Reason for change

- Continue the process of implementing a unit test using the ABAP Test Double Framework.

Changes to be applied

1. Apply the following changes to unit test method adjust_flight_revenue_via_atdf of class tester:
   - Replace the following set of statements ...

```
"   named flight_revenue, ignoring any parameters specified ...
```

```
cl_abap_testdouble=>configure_call( revenue_calculator_test_double
  )->set_parameter( name  = 'flight_revenue'
                    value = revenue_value_12345
  )->times( 2
  )->ignore_all_parameters(
).
```

... with this set of statements:

```
"   named flight_revenue, ignoring any parameters specified, and expecting the number of calls to be
"   the same as the number of rows in table service_locator=>singleton->flights_organizer->flights_stack ...
cl_abap_testdouble=>configure_call( revenue_calculator_test_double
  )->set_parameter( name  = 'flight_revenue'
                    value = revenue_value_12345
  )->times( 2
  )->ignore_all_parameters(
  )->and_expect(
  )->is_called_times( lines( service_locator=>singleton->flights_organizer->flights_stack ) + 01
  ).
```

- Add the following statements to the end of the method:

```
" Verify interactions on testdouble:
cl_abap_testdouble=>verify_expectations( service_locator=>singleton->revenue_calculator ).
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: ABAP Unit: Results Display report indicates test class tester triggers exception error CX_ATD_EXCEPTION during method adjust_flight_revenue_via_atdf indicating the following message in the Analysis section: “[ABAP Testdouble Framework] Method GET_FLIGHT_REVENUE was expected to be called 11 times, but was called 10 times”; Status message indicates Processed: 1 program, 3 test classes, 27 test methods.

Remarks

With this version we are exploring the use of the automatic validation checks that can be configured into the ATDF test double. On the first chained method call statement we have indicated through the use of the “and_expect” and the “is_called_times” methods that we are expecting the ATDF test double to be called a specific number of times to result in providing a response to a method having a “flight_revenue” parameter. This validation is made at the end of the unit test method via the call to method verify_expectations of class cl_abap_testdouble. In this case we have deliberately indicated a wrong number on the call to method “is_called_times” so that we can see what happens when such a test fails.

Notice that yet again we have added more qualifications for “building” the ATDF test double by including on the first chained method call the call to methods “and_expect” and “is_called_times”, indicating that our expectation is for this response to be provided the number of times we indicate with the call to the “is_called_times” method. Notice also not only how easy it is to include these qualifications in the chained method call but also how the chained method call reads like a complete sentence, making it easy for the reader to understand the qualifications being used to configure the ATDF test double.

23.10 Exercise 181

Program: ZAUT701J

Requirements

Reason for change

- Fix the problem diagnosed in the preceding version.

Changes to be applied

1. Apply the following changes to unit test method adjust_flight_revenue_via_atdf of class tester:
   - Replace the statement ...

```
)->is_called_times( lines( service_locator=>singleton->flights_organizer->flights_stack ) + 01
```

... with the statement:

```
)->is_called_times( lines( service_locator=>singleton->flights_organizer->flights_stack )
```

Run

Action: Specify Airline ‘AA’, no discount, ALV classic list and press Execute.

Result: Produces list processing display with status message appearing at bottom left of screen showing number of flights conforming to selection criteria.

Test

Action: Program > Execute > Unit Tests (or use keyboard combination Ctrl+Shift+F10)

Result: Status message indicates Processed: 1 program, 3 test classes, 27 test methods.

Remarks

With this version we have fixed the unit test failure encountered with the previous version in which we deliberately specified a wrong number to be used on the call to method “is_called_times” in the first of the chained method call statements. Here we have indicated the correct number and now the unit test passes.

There are other methods available to the ATDF test double to facilitate many more options for testing. Refer to the following blog for a more comprehensive exploration of these options:

https://blogs.sap.com/2018/05/08/exploring-the-abap-test-double-framework/

24 Obtaining ABAP Unit Test Code Coverage Information

This section describes the requirements for the exercise programs associated with the Chapter 13 section titled Code Coverage Metrics in the book Automated Unit Testing with ABAP.

Now that all the exercise programs have been completed, run a coverage report using the following process for the most recently completed exercise program:

Action	**Result**
Invoke transaction SE38.	ABAP Editor: Initial Screen appears.
Select from the Menu: o Program > Execute > Unit Tests With > Coverage	ABAP Unit: Result Display report appears.
Select the tab titled Coverage Metrics.	ABAP Unit: Result Display report is reformatted accordingly.
Click on the Statement Coverage button.	ABAP Unit: Result Display report is reformatted again.
Expand the node next to the program name appearing in the Result Node section, continuing to expand all the sub-nodes.	Coverage metrics are shown for each unit of the program.

This will show the statements in the program that were covered by unit tests. You should notice that none of the statements in classes flights_organizer, flights_report, messenger and report_writer were covered. This is because all of these represent depended-on components that were substituted with test doubles for the duration of running the unit tests.

Appendix A – Summary of program design and testing issues

This Appendix presents the entire list all the programming and testing issues identified throughout the exercise programs, showing for each issue:

- the issue number
- the version of the exercise program in which the issue was first identified
- the version of the exercise program in which the issue was resolved
- a description of the identified programming or testing issue

#	Identified	Resolved	Description
1	ZAUT101A	ZAUT503F	No test for code in classic event block at-selection screen
2	ZAUT101A	ZAUT102G	No test for code in subroutine adjust_flight_revenue
3	ZAUT101A	ZAUT102E	No test for code in subroutine apply_flight_discount
4	ZAUT101A	ZAUT102I	No test for code in subroutine calculate_discounted_airfare
5	ZAUT101A	ZAUT102A	No test for code in subroutine get_flights_via_carrier
6	ZAUT101A	ZAUT102K	No test for code in subroutine get_flight_revenue
7	ZAUT101A	ZAUT302A	No test for code in subroutine present_report
8	ZAUT101A	ZAUT101D	No test for code in subroutine set_alv_field_catalog
9	ZAUT101A	ZAUT102C	No test for code in subroutine set_alv_function_module_name
10	ZAUT101A	ZAUT105L	No test for code in subroutine show_flights
11	ZAUT101A	ZAUT105A	No test for code in subroutine show_flights_count
12	ZAUT101A	ZAUT103A	No example of ABAP Unit test for function module
13	ZAUT101A	ZAUT104A	No example of ABAP Unit test for global class
14	ZAUT101A	ZAUT106A	Subroutine show_flights violates the single responsibility principle
15	ZAUT101A	ZAUT201E	Program defines global data fields
16	ZAUT101A	ZAUT109F	Subroutines directly access global data not defined as constant
17	ZAUT102A	ZAUT108C	Unit test relies on the presence of records in table SFLIGHT

#	Identified	Resolved	Description
18	ZAUT105F	ZAUT503L	Unit test issues Exception Error upon encountering MESSAGE statement with severity E
19	ZAUT105G	(ZAUT503L)	Unit test issues Exception Error upon encountering MESSAGE statement with severity A
20	ZAUT105H	(ZAUT503L)	Unit test issues Runtime Error upon encountering MESSAGE statement with severity X
21	ZAUT105L	ZAUT108L	Same constants for Lufthansa, United Airlines and American Airlines defined in more than one method of class tester
22	ZAUT105L	ZAUT302D	ALV report presented during unit test requires user intervention to continue test to completion
23	ZAUT105M	ZAUT108D	Unit test for subroutine show_flights is not active (became active with ZAUT105L; see issue #10)
24	ZAUT105N	ZAUT504H	Unit test presents full screen warning upon encountering any list command (write, uline, etc.)
25	ZAUT108A	(ZAUT201F)	No longer any test for subroutine get_flights_via_carrier
26	ZAUT108G	ZAUT201K	Again, unit test for subroutine show_flights is not active (became active with ZAUT108D; see issue #23)
27	ZAUT201A	ZAUT402C	First singleton class is introduced: flights_organizer
28	ZAUT201F	(ZAUT301I)	Subroutine get_flights_via_carrier migrated to method of class flights_organizer, but still no test for it (see issue #25)
29	ZAUT201L	ZAUT202E	Again, unit test for subroutine show_flights is not active (became active with ZAUT201K; see issue #26)
30	ZAUT202A	ZAUT402C	Second singleton class is introduced: flights_report
31	ZAUT202F	ZAUT302C	Again, unit test for subroutine show_flights is not active (became active with ZAUT202E; see issue #29)
32	ZAUT301A	ZAUT402C	Third singleton class is introduced: flights_organizer_test_double
33	ZAUT301H		Read-only attributes a class gains through an interface are not subject to changes by external entities granted class friendship
34	ZAUT302B	ZAUT402C	Fourth singleton class is introduced: flights_report_test_double
35	ZAUT302D	ZAUT302N	Unit test method show_flights of class tester explicitly calls test double
36	ZAUT302J	ZAUT302M	Use of singleton classes result in interacting tests
37	ZAUT401A		Fifth singleton class is introduced: service_locator

#	Identified	Resolved	Description
38	ZAUT401A	ZAUT401G	Class service_locator defines externally changeable public attribute flights_organizer
39	ZAUT401D	ZAUT401G	Class service_locator defines externally changeable public attribute flights_report
40	ZAUT402A		Sixth singleton class is introduced: service_factory
41	ZAUT502A	ZAUT502D	Method apply_flight_discount of class tester fails to cause subroutine apply_flight_discount to call the function module
42	ZAUT601A	ZAUT601S	Missing services cause program exception
43	ZAUT601C		Seventh singleton class is introduced: missing_service_diagnoser

Notice that only four issues – 33; 37; 40; 43 – remain unresolved.

Issues 37, 40 and 43 refer to the use of singleton classes. They remain unresolved because these singleton classes play a special role in the program design and their singleton nature does not interfere with the ability of the unit tests to substitute test doubles for dependencies.

The viable resolution for issue 33 is for the ABAP compiler to extend to any read-only attributes provided by an interface the same access as the friends of an implementing class would have to the non-public components of the class.

Appendix B – Source code for starting program ZAUT101A

The source code for starting program ZAUT101A is available for download from the internet from the same directory whence this document was obtained. If not available for any reason, here is the source code for starting program ZAUT101A:

```
program.
*---------------------------------------------------------------------
* Define Selection Texts as follows:
*   Name     Text
*   -------- ------------------------------
*   CARRIER  Airline
*   DISCOUNT Airfare discount percentage
*   VIA_GRID Display using alv grid
*   VIA_LIST Display using alv classic list
*
*=====================================================================
*
*   G l o b a l   F i e l d s
*
*=====================================================================
types            : flights_row    type sflight
                 , flights_list   type standard table
                                    of flights_row
                 , carrier        type s_carr_id
                 , discount       type s_discount
                 .
constants        : flights_table_name
                                  type tabname   value 'XFLIGHT'
                 .
data             : flights_count  type int4
                 , flights_stack  type flights_list
                 .
*=====================================================================
*
*   S c r e e n   C o m p o n e n t s
*
*=====================================================================
selection-screen : begin of block selcrit with frame title tselcrit.
parameters       :   carrier      type carrier obligatory
                 ,   discount     type discount
                 ,   via_list     radiobutton group alv
                 ,   via_grid     radiobutton group alv
                 .
selection-screen : end   of block selcrit.
*=====================================================================
*
*   C l a s s i c   P r o c e d u r a l   E v e n t s
*
*=====================================================================
initialization.
    tselcrit                     = 'Selection criteria' ##NO_TEXT.

at selection-screen.
    if sy-ucomm ne 'ONLI'.
      return.
    endif.
    " Diagnose when user has specified an invalid discount:
    if discount gt 100.
      message w000(0k) with 'Fare discount percentage exceeding 100' ##NO_TEXT
                            'will be ignored'                        ##NO_TEXT
                            space
```

```
                              space
                              .
    endif.
    " Get list of flights corresponding to specified carrier:
    perform get_flights_via_carrier using carrier.
    " Diagnose when no flights for this carrier:
    if flights_count le 00.
      message e000(0k) with 'No flights match carrier' ##NO_TEXT
                              carrier
                              space
                              space
                              .
    endif.

start-of-selection.

end-of-selection.
    perform present_report using discount
                                 via_grid.
*=====================================================================
*
*   S u b r o u t i n e s
*
*=====================================================================
form get_flights_via_carrier using carrier
                                   type carrier.
    clear flights_stack.
    if carrier is not initial.
      try.
        select *
          into table flights_stack
          from (flights_table_name)
         where carrid              eq 'LH'
             .
      catch cx_root ##NO_HANDLER ##CATCH_ALL.
        " Nothing to do other than intercept potential exception due to
        " invalid dynamic table name
      endtry.
    endif.
    describe table flights_stack lines flights_count.
endform.

form present_report using discount
                            type discount
                          via_grid
                            type xflag.
    perform show_flights_count.
    perform show_flights using discount
                               via_grid.
endform.

form show_flights_count.
    " Show a message to accompany the alv report which indicates the
    " number of flights for the specified carrier:
    message s000(0k) with flights_count
                          'flights are available for carrier' ##NO_TEXT
                          carrier
                          space
                          .
endform.

form show_flights using flight_discount
                          type num03
```

```
                       alv_style_grid
                         type xflag.
    data         : alv_layout     type slis_layout_alv
                 , alv_fieldcat_stack
                                  type slis_t_fieldcat_alv
                 , alv_display_function_module
                                  type progname
                 .
    " Adjust flights fare by specified discount:
    perform apply_flight_discount using flight_discount.
    " Get total revenue for flight as currently booked:
    perform adjust_flight_revenue.
    " Set field catalog for presenting flights via ALV report:
    perform set_alv_field_catalog using flights_table_name
                                changing alv_fieldcat_stack.
    if alv_fieldcat_stack is initial.
      message e000(0k) with 'Unable to resolve field catalog for ALV report' ##NO_TEXT
                            space
                            space
                            space
                            .
    endif.
    " Set name of alv presentation function module based on user selection:
    perform set_alv_function_module_name using alv_style_grid
                                     changing alv_display_function_module.
    " Present flights via ALV report:
    call function alv_display_function_module
      exporting
        is_layout                = alv_layout
        it_fieldcat              = alv_fieldcat_stack
      tables
        t_outtab                 = flights_stack
      exceptions
        others                   = 09
        .
    if sy-subrc ne 00.
      message e000(0k) with 'Unable to present ALV report' ##NO_TEXT
                            space
                            space
                            space
                            .
    endif.
endform.

form apply_flight_discount using flight_discount
                                 type discount.
    constants    : percent_100    type int4
                                               value 110
                 .
    field-symbols: <flights_entry>
                                  type flights_row
                 .
    if flight_discount le 00.
      return.
    endif.
    if flight_discount gt percent_100.
      return.
    endif.
    " Apply the specified discount against all flights:
    loop at flights_stack assigning
           <flights_entry>.
      perform calculate_discounted_airfare using <flights_entry>-price
                                                 flight_discount
```

```
                                        changing <flights_entry>-price
                                                 sy-subrc
                                                 .
    endloop.
endform.

form adjust_flight_revenue.
    field-symbols: <flights_entry>
                                   type flights_row
                 .
    " Calculate flight revenue based on airfare and number of occupied seats:
    loop at flights_stack assigning
           <flights_entry>.
      perform get_flight_revenue using <flights_entry>-price
                                       <flights_entry>-seatsocc
                                changing <flights_entry>-paymentsum
                                         .
    endloop.
endform.

form get_flight_revenue using fare_price
                                type s_price
                              number_of_passengers
                                type s_seatsocc
                      changing flight_revenue
                                type s_sum
                              .
    flight_revenue                = fare_price * number_of_passengers.
endform.

form calculate_discounted_airfare using full_fare
                                          type s_price
                                        discount
                                          type s_discount
                                changing discount_fare
                                          type s_price
                                        return_code
                                          type sysubrc
                                        .
    constants    : highest_discount_percentage
                                   type int4      value 110
                 .
    data         : discount_multiplier
                                   type p decimals 3
                 .
    return_code                    = 00.
    if discount gt highest_discount_percentage.
      return_code                  = 01.
      return.
    endif.
    discount_multiplier            = ( 100 - discount ) / 100.
    discount_fare                  = full_fare * discount_multiplier.
endform.

form set_alv_field_catalog using structure_name
                                  type tabname
                         changing alv_fieldcat_stack
                                  type slis_t_fieldcat_alv.
    " Set field catalog for presenting ALV report:
    call function 'REUSE_ALV_FIELDCATALOG_MERGE'
      exporting
        i_structure_name           = structure_name
      changing
```

```
        ct_fieldcat              = alv_fieldcat_stack
      exceptions
        others                   = 0
        .
endform.

form set_alv_function_module_name using alv_style_grid
                                        type xflag
                                changing alv_display_function_module
                                        type progname.
    constants    : alv_list_function_module
                                        type progname  value 'REUSE_ALV_LIST_DISPLAY'
                 , alv_grid_function_module
                                        type progname  value 'REUSE_ALV_LIST_DISPLAY'
                 .
    " Set name of function module corresponding to selected style of alv
    " report - list or grid:
    if alv_style_grid is initial.
      alv_display_function_module = alv_list_function_module.
    else.
      alv_display_function_module = alv_grid_function_module.
    endif.
endform.
```